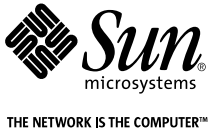

Project Neptune

Programmer's Reference Manual (Open Form)

Revision: 1.4.2o Oct 10 2007

Sun Microsystems, Inc.
4150 Network Circle
Santa Clara, CA 95054
U.S.A. 650-960-1300

Products Rights Notice:

You understand that these materials were not prepared for public release and you assume all risks in using these materials. These risks include, but are not limited to errors, inaccuracies, incompleteness and the possibility that these materials infringe or misappropriate the intellectual property right of others. You agree to assume all such risks.

THESE MATERIALS ARE PROVIDED BY THE COPYRIGHT HOLDERS AND OTHER CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS (INCLUDING ANY OF OWNER'S PARTNERS, VENDORS AND LICENSORS) BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THESE MATERIALS, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Sun, Sun Microsystems, the Sun logo, Solaris, OpenSPARC T1, OpenSPARC T2 and UltraSPARC are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. in the U.S. and other countries. Products bearing SPARC trademarks are based upon architecture developed by Sun Microsystems, Inc. UNIX is a registered trademark in the U.S. and other countries, exclusively licensed through X/Open Company, Ltd. The Adobe logo is a registered trademark of Adobe Systems, Incorporated. Part of the products covered by these materials may be derived from the Berkeley BSD systems licensed by the University of California. Sun Microsystems, Inc. has intellectual property rights relating to technology embodied in the product described in these materials. This distribution may include materials developed by third parties who have intellectual property rights therein. Products covered by and information contained in these materials may be controlled by U.S. Export Control laws and may be subject to the export or import laws in other countries. Nuclear, missile, chemical biological weapons or nuclear maritime end uses or end users, whether direct or indirect, are strictly prohibited. Export or reexport to countries subject to U.S. embargo or to entities identified on U.S. export exclusion lists, including, but not limited to, the denied persons and specially designated nationals lists may be prohibited.

Contents

75

List of Figures

List of Tables

Preface

0.1 Introduction and Scope

The Sun Microsystems Project Neptune, is a feature-rich, high performance PCI Express 1.1 compliant Dual 10Gbps Ethernet / Quad 1G bps RGMII Network Interface Chip.

This Document is intended to be used as a guide for driver and firmware development as well as to provide a software-hardware view of the Neptune Chip.

0.2 Overview

This Document includes architectural and functionality description along with register definition and programming information in each chapter. Its organization mirrors the Chip's organization as well as its relationship with software. The Chip is roughly divided into these sections:

PCI Express/Boot Subsystem includes

1. SPROM and EPROM boot units
2. PCI Express Unit (PEU)

Network Interface Core Unit (NIU) Subsystem includes

1. Receive and DMA Subsystem
2. Transmit and DMA Subsystem
3. Ethernet Subsystem

0.3 Glossary

block A contiguous range (for example, 8 Kbyte, 32 Kbyte) of memory location.For Neptune, it is a block of contiguous I/O addresses. A block may be used to hold one or more packet buffers of the same size, or when used to store jumbo frames, multiple blocks are used to build a packet buffer.

logical address An address within a logical page.

logical condition (LC) A condition that, when true, may ultimately trigger an interrupt. It may be logically "level" in that the condition is constantly being evaluated, or it may be logically "edge triggered" in that a state is maintained when it first occurred. This state needs to be cleared to enable the hardware to detect the next occurrence.

logical device (LD) A term used generically to refer to a functional block that may ultimately cause an interrupt. This may be a transmit DMA channel, a receive DMA channel, a MAC, or other system-level blocks. Within a logical device, one can define one or more logical conditions. Each logical condition may be masked. A logical device may have up to two groups of logical conditions. Each group will have one "summary" flag (LDF). Depending on the logical conditions captured by the group, this flag may be level or may be edge-triggered. An unmasked logical condition, when true, *may* trigger an interrupt.

logical device flag (LDF) A logical **or** of some LCs within the LD. There are up to two flags per LD.

logical device group (LDG) A group of logical devices sharing an interrupt. A group may have only one LD. Currently, we support up to 64 logical device groups.

logical device group interrupt (LDGI) The interrupt associated with a LDG. This interrupt is controlled by a one-shot mechanism, that is, hardware will issue only one single interrupt, and software needs to arm the LDG again to enable it to issue another interrupt.

logical device interrupt mask (LDGIM) A per-LD mask that defines when the LC becomes true, whether a device may issue an interrupt.

logical device group state mask (LDGSM) A per-LDG mask that defines which part of the LDSV is visible to the LDG. This also defines which LD is part of a LDG.

logical device state vector (LDSV) A read-only state vector capturing the LDFs of *all* the LDs.

logical page A contiguous range of memory location. If an address posted by software is within a logical page, it will be translated to a physical address by replacing the base address of the logical page with the base address of the physical page. The size of the logical page is programmable.

receive block ring (RBR) A ring buffer of memory blocks posted by software.

receive completion ring (RCR) Stores the addresses of the buffers used to store incoming packets.

receive DMA channel (RDC) Consists of an RBR, an RCR, and a set of control and status registers. A receive DMA channel is selected after an incoming packet is classified. A packet buffer is derived from the pool and used to store the incoming packet. Each channel is capable of issuing interrupt to software based on the queue length of the receive completion ring or a timeout.

receive DMA channel table (RDC table) A table of 16 entries. Each entry contains one RDC. Each table defines the group of RDCs an incoming packet can be deposited into. The classification hardware will choose a table as an intermediate step before a final RDC is selected. The 0th entry of the table is the default RDC. This default RDC is used to queue error packets within the group. This default can be one of the RDC(s) in the group.

receive DMA channel group (RDC Group) A group of receive DMA channels stored in a corresponding RDC table.

receive packet classification Incoming packets will be classified based on layer 2/3/4 information. The result will select a Receive DMA Channel to store the packet.

system interrupt The Interrupt sent to the CPU. In UltraSPARC T2, this signal is sent to the NCU which will look up the (CPU ID, Interrupt Number) pair. In Neptune, this is the PCI-Ex interrupt. Depending on the setting in the config space, the appropiate interrupt type will be issued.

transmit ring (TR) The data structure built-in system memory for software to post transmission requests.

transmit DMA channel (TDC) Consists of a transmit ring and a set of control and status registers. Software posts packets as a lists of gather pointers. Completion of packet transmission is indicated in status registers. Software needs to poll the status registers or enable hardware to issue interrupt after a specific packet is transmitted.

0.4 Revision History

File Revision History

Date	Revision ID	Comment
01/19/07	1.4	Update organization and format
04/12/07	1.4.1	Add External IP conditions
10/10/07	1.4.2	Added Open Form

CHAPTER 1

Neptune Applications

1.1 Basic Neptune Application space

Neptune is a shared, virtualized, non-blocking, multi-homed networking PCI Express compliant ASIC. It is supported along with multi-OS device drivers and boot firmware for the Atlas family of network interface cards and server motherboards. Its networking core function is also instantiated in Sun's CMT processors.

The Neptune architecture and implementation is specifically designed to exploit multicore and multithreaded processors with minimum CPU load while maximizing network I/O throughput.

Its main features include: quad multi-speed Ethernet MACs (two of them up to 10Gb/s), line rate Layer 2/3/4 ingress packet classification (up to 33Mpkts/s), packet movement using multiple (up to 40) DMA engines, and virtualization / partitioning with per function or per DMA channel granularity. Detailed features are listed in the next Chapter

Neptune's four Ethernet MACs support motherboards with six physical interfaces: two 10Gigabit on XFI modules plus four 1000BASE-T RJ-45. This provides the standard quad Gigabit configuration at a low cost, and allows for 10Gigabit optical modules to be inserted for 10Gigabit operation. Only four of the six interfaces are used at any given time.

Neptune's 10Gigabit Ethernet MAC also supports direct connection to the ATCA extended fabric, where four XAUI lanes are driven at 3.125Gbps each if the ATCA switch supports 10Gigabit Ethernet, or a single lane is driven at 1.25Gbps if the switch is just a 1Gigabit Ethernet device. The other two Neptune ports are available for connection to the RTM and front panel Ethernets, or to the base fabric. Neptune does not directly provide a netconsole serial over Ethernet interface.Text.

CHAPTER 2

Neptune Introduction

2.1 Features

- Line rate packet classification
  - ~33 MPkts/sec
  - L1 - L4 IPv4/v6 header parsing
  - Hierarchial classification : packet classes, VLAN (4K per port), ternary matches (256 entries) and hash function
  - 16 unique MAC addresses per port
- TCP/UDP/IP checksum offload
- Jumbo frame support (upto 9216B)
- Multiple DMA engines
  - Tx - 24 with DRR/gather support; Rx - 16 with WRED support
  - Flexible binding between DMAs and ports
  - Support CPU/thread affinity (avoid context switching)
- Virtualization support
  - DMA resource separation by logical groups (up to 8 partitions)
  - Descriptor address relocation
- Interrupts
  - Interrupt coalescing
  - Mailbox
  - INTX, MSI, MSI-X
- Multi-speed ports
  - 2 quad-speed (10M/100M/1G/10G)
  - 2 triple-speed (10M/100M/1G)
  - 2x XAUI for 10Gb ports
  - 4x RGMII for 10M/100M/1G ports
- PCI Express 1.1
  - x1/x4/x8
  - Support up to 4 functions - flexible binding between DMAs and functions
  - Support transaction timeout and various error handling
  - Relaxed ordering memory access with on-chip re-ordering - 32 outstanding transactions

2.2 Functional Overview

A high-level block diagram is shown in FIGURE 2-1.

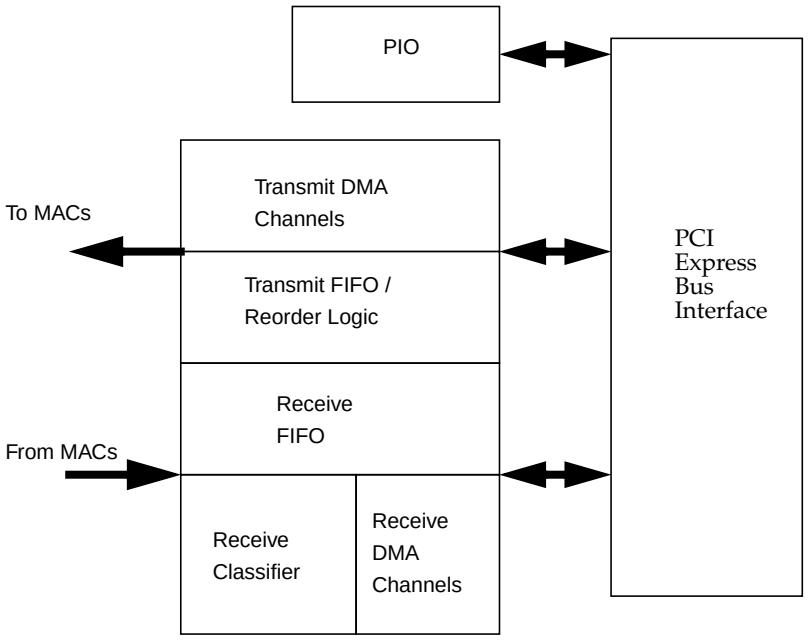

FIGURE 2-1 Neptune Architecture

Neptune is a PCI Express multi-function device. The Program I/O (PIO) module is where memory-mapped I/O loads and stores to CSRs are dispatched to different functional units. The MACs are the Ethernet controllers, supporting the link protocol and statistics collection. Packets received are first classified based on the packet header information. The classification result determines the receive DMA channel. Transmit packets are posted into a transmit DMA channel. Each packet is made up of a gather list. Hardware supports checksum offload, on both receive and transmit data.

2.2.1 Resource Grouping And Virtualization Support

A general definition and usage of virtualization is beyond the scope of this document. For the purpose of this document, we concern ourself with accessing the device via PIOs and the address used in read/write requests. The latter relates to memory protection. Together these two features enable resource (both memory and DMAs) isolation, which is the basis of virtualization. Note that Neptune is not a PCI-IOV compliant device. The usage here is for software to isolate accesses from different OS images within a single hardware system. This will require the support of non-standard software. Not all hardware resources support virtualization directly. In some cases, it is to simplify the design, in others, it is the fundamental physical limit (for example, the MAC hardware state machine). Software driver access to these hardware blocks in an environment where virtualization is supported can be achieved via the virtualization management software layer.

In Neptune, the hardware is organized as a four-function device. Each function supports three BARs. One BAR is dedicated for PCI MSI-X. Within each function, two additional address ranges (or two separate BARs) are defined: one for management one for virtualization. The entire device may be accessed through the management addresses. Virtualization addresses, on the other hand, only have accesses to a set of defined DMAs. The CSRs of multiple DMA channels can be grouped into an 8KB page within the virtualization address ranges. The grouping itself is defined by a table in the management address range.

To support memory protection, each transmit or receive DMA supports two logical pages. The addresses in the configuration registers, packet gather list pointers on the transmit side, and the allocated buffer pointer on the receive side will be relocated accordingly. The logical page registers are only accessable via the management address ranges.

In Neptune, system software may only expose the virtualization BAR region to the driver software, enabing the driver software to control the DMAs via PIOs. In addition, system software may also defines the logical page registers for these DMAs via the management BAR, which limits the addresses in the descriptors the driver posted. Together, this enforces DMA and memory resource the driver software may use.

Note that the datapath of a packet, as described in Section 2.2.4, is not affected by these features. The following figure summarizes the usage supported.

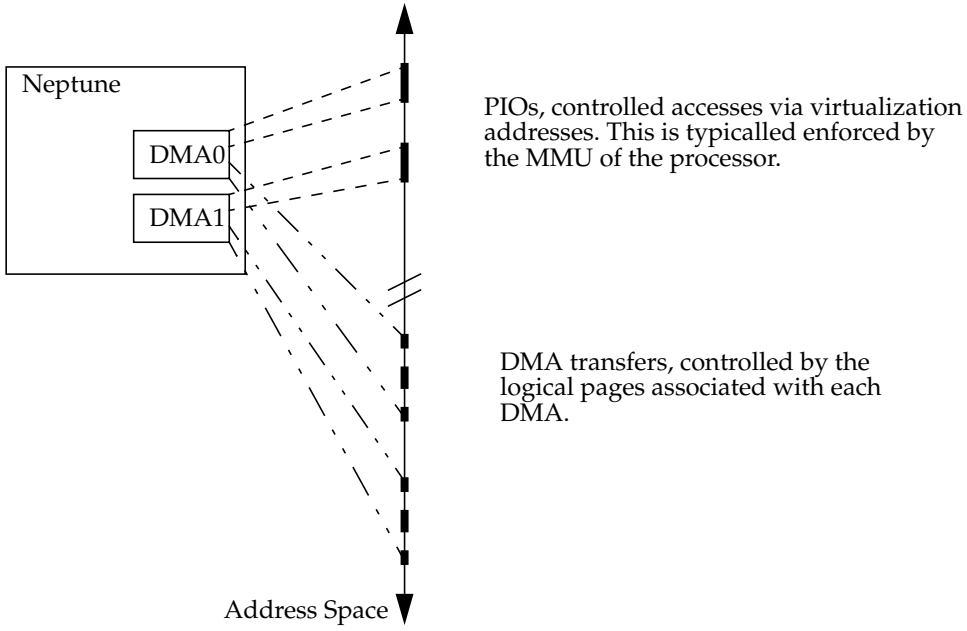

FIGURE 2-2 Virtualization support.

2.2.2 Interrupt Hierarchy

To support the sharing of available system interrupts, which may be less than the number of logical devices (LDs), LDs are grouped into logical device groups (LDGs). The state of the LDs that are part of an LDG may be read by software. Not all LDs belonging to a group can trigger an interrupt. This is controlled by the LD's interrupt mask. For example, a transmit DMA channel may be part of an LDG, and software may examine the flags associated with the transmit DMA by setting the LD's LDG number. However, the DMA will not trigger an interrupt if the corresponding bit in the interrupt mask is not asserted.

Associated with an LDG is a system interrupt control comprising an arm bit, a timer, and system interrupt data. System interrupt data is the data associated with the system interrupt and may be used to select the final interrupt sent to the processor. Driver software writes to the register to set the arm bit to 1 and to set the value of the timer. Hardware will start counting down the timer. An interrupt will only be issued if the timer is zero, the arm bit is set, and one or more LD's in LDG, have their flags set and not masked. This ensure that there is a period of "quiet" time between interrupt services.

Software needs to clear the state or adjust the conditions of individual LD after servicing. Note that hardware does not support any aggregate updates applied to the entire LDG.

For Neptune, the higher order two bits of System Interrupt Data is used to select the PCI function, and the lower order five bits will be used if either MSI or MSI-X is enabled. For MSI, these bits forms part of the MSI data, and for MSI-X, these bits are used to select a particular MSI vector.

2.2.3 PIO and Datapath Interfaces

For Neptune, the PCI Express interface will support a System Interrupt Data to interrupt translation. Depending on how the PCI configuration space is set up, either an INT_x or a MSI will be issued.

Internal to the PIO unit, a FIFO is used to queue up requests from CPUs. Requests will be read one by one and dispatched to the different functional units. Write requests may be dispatched to the functional unit so long as the functional unit can accept the request. Before a read request can be issued, *all* prior requests (read or write) have to be acknowledged.

2.2.4 Life of a Packet

A description of the receive path follows.

A more detailed block diagram of the receive classification logic and receive FIFO is shown in FIGURE 2-3.

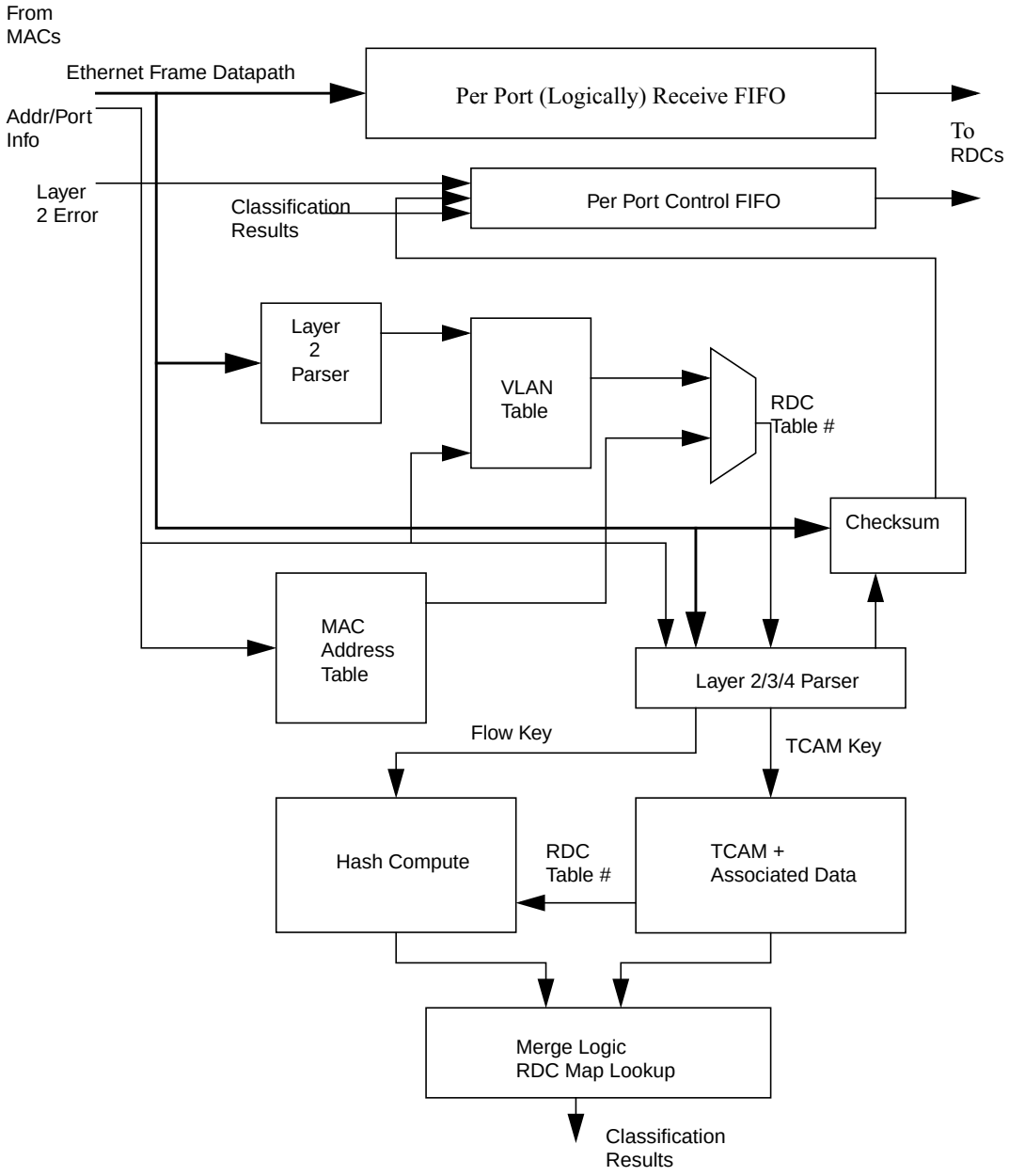

FIGURE 2-3 Receive Buffer and Classification Logic

MACs are the Ethernet media access controllers that support the Ethernet protocol. They contain the Layer 2 protocol logic, statistic counters, address matching, and filtering logic. The output from the MACs contains information on the destination address, whether it is one of the programmed individual addresses or an accepted group address, and the index associated with the address in that category.

Frames from different physical ports are stored temporarily in a per-port receive FIFO. While they are being stored into the FIFO, the frame will also be copied to the packet classification and checksum engines. The classification logic will determine which RDC group the packet belongs to and an offset into the RDC Table where the final RDC is determined. There are a total of eight RDC Tables.

The Layer 2 parser processes the Ethernet header to determine if the received frame contains a VLAN tag or LLC/SNAP header. For a VLAN-tagged packet, the VLAN ID is used to look up a VLAN table to determine the RDC table number for the packet. Hardware will also look up the MAC address table to determine an RDC table number based on the destination MAC address information. Software can program which of the two groups to use in subsequent classification. The output of the Layer 2 parser together with the resulting RDC table number will be passed to the Layer 2/3/4 parser.

The Layer 2/3/4 parser will examine the ethertype, tos/dscp field, and the protocol id/next header field to determine if the IP packet needs further classification. It is hardwired to recognize some fixed protocol such as TCP or UDP. It also supports a number of programmable Protocol IP numbers. If the packet needs further classification, it will generate a flow key and a TCAM key.

The TCAM key is sent to the TCAM unit for an associative search. If there is a match, the result may override the RDC table selection from L2 and/or contain an offset into the Layer 2 RDC table and ignore the result from the hash unit. The outputs from the hash unit and the TCAM unit will be merged to determine the RDC. This RDC is used to store the packet.

The outputs of the classification unit are stored into the Control FIFO.

Hardware supports checksum offload and will simply compare the calculated values with the values embedded in the frame. The result will be sent to software through the completion status. No discard decision is made. Note that checksum errors do not affect the L3/4 classification logic. Similarly, the error status will be sent to software through the completion status.

The Receive FIFO is logically organized per physical port. Layer 2/3/4 error information has to be logically synchronized with the classification result of the corresponding frame.

Logically, 16 receive DMA channels are available to incoming packets. The datapath engine is common across all DMA operations. It is also used to prefetch the receive blocks or update the completion ring of the RDCs. FIGURE 2-4 illustrates the functional blocks within the receive DMA channel logic block.

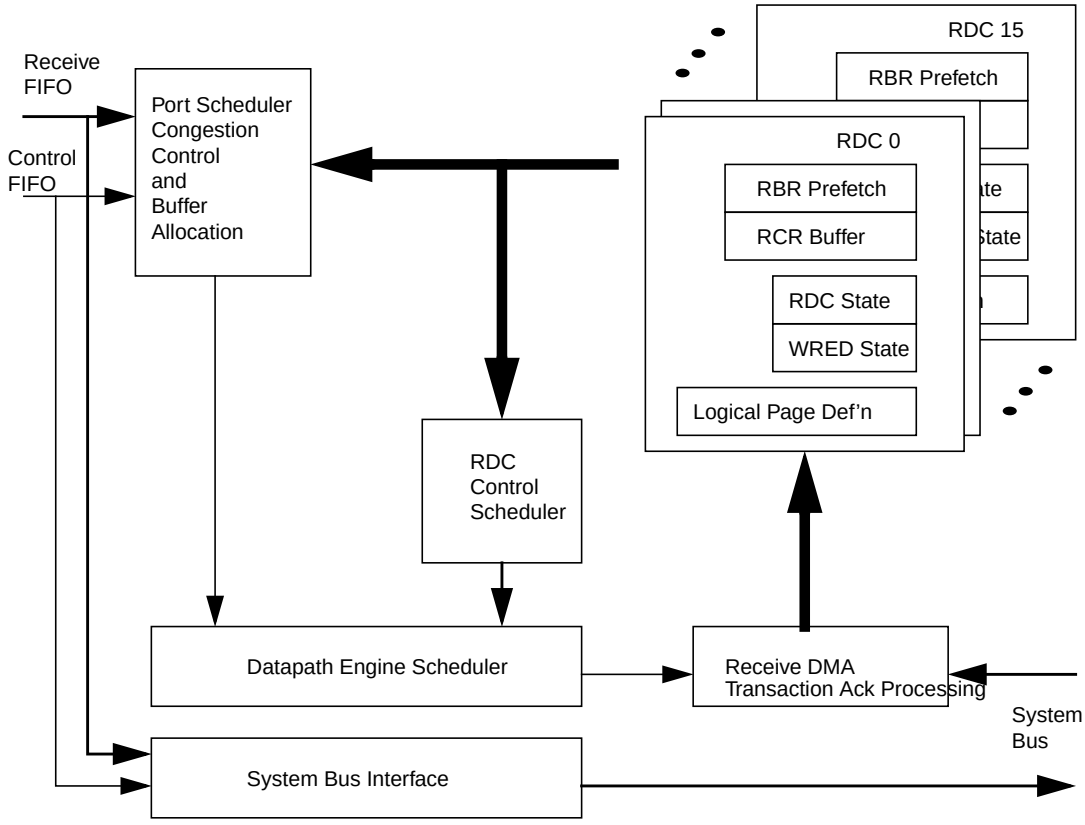

FIGURE 2-4 Receive DMA Channels

Each receive DMA channel (RDC) has a receive block ring (RBR), a receive completion ring (RCR), and the state associated with the RDC. Physically, they are allocated as ring buffers in DRAM. To support partitioning, each RDC supports two logical pages. All the addresses posted by software, such as the configuration of the ring buffers and buffer block addresses, are translated to physical addresses when used to reference system memory.

Software posts buffer blocks into the RBR. The size of each block is programmable, but fixed per channel. Software can specify up to three packet sizes for hardware to partition a block. Each block can only contain packet buffers of the same size.

To reduce the per-packet overhead, hardware maintains a prefetch buffer for the RBR and a tail buffer for the RCR. When the RBR prefetch is low, a request will be issued to the DRAM system to retrieve a cache of block addresses from the ring. Or

if the RCR tail buffer needs to be updated, a write request will be issued. The consistency of the RCR state is maintained by the hardware. The RDC control scheduler will maintain the fairness among the RDCs.

The port scheduler examines whether any frames are available from the Receive FIFO and the Control FIFO, and decides which port to service first. A deficit round robin scheduler is implemented. From the control header, the scheduler determines which RDC to check for congestion and retrieves a buffer to store the frame. Congestion is determined by a WRED algorithm applied on the receive completion ring. If the RDC is not congested, a buffer address is allocated according to the packet size. Packet data requests are issued as posted writes.

The datapath engine will fairly schedule the requests from the port scheduler and the RDC control scheduler and issue the requests to the DRAM.

The RCR buffer will be updated after issuing the write requests for the entire packet. The DMA status registers will be updated every time the RCR buffer is updated. Software may poll the DMA status registers to determine if any packet has been received. When the RCR queue length reaches a threshold or a timeout occurs, hardware may update the RCR buffer and at the same time, write the DMA status registers to a software defined mailbox. The software state will then be updated, and a logical device flag may be raised. The LDF may then lead to a system interrupt. Hardware will maintain the consistency of the DMA status registers and the RCR in that the status registers reflects the content of the RCR in system memory.

There are 24 Transmit DMA Channels in Neptune. FIGURE 2-5 shows the logical view of the transmit hardware. For Neptune, four ports are supported. Each channel consists of a transmit ring and a set of control and status registers. Similar to the receive side, each channel supports two logical pages. Addresses in the transmit ring and configuration registers are subjected to a translation to convert to physical addresses.

The transmit ring is built from a ring buffer in system memory. Software posts packets into the transmit ring and signals the DMA hardware that packets have been queued. Each packet is built as a gather list. (Note that hardware will check to ensure that the total transfer size associated with a packet does not exceed the maximum hardware limit. This includes the internal headers.) When the transmit ring is not empty, hardware will prefetch the transmit ring into a per channel buffer.

Any DMA channel can be bound to one of the Ethernet ports by software. This is controlled by a mapping register at the per-port DRR scheduler. The DRR scheduler may switch to a different channel on packet boundary. This guarantees there will be no packet interleaving from different DMA channels. The scheduler will first acquire an available buffer for that port. If it is available, a memory request will be issued. A buffer tag identifying the buffer is needed to reorder potentially out-of-order read returns. This tag is linked to the request/ACK ID.

The Ethernet ports are serviced in round-robin order, and requests from different ports may be interleaved. See FIGURE 2-5.

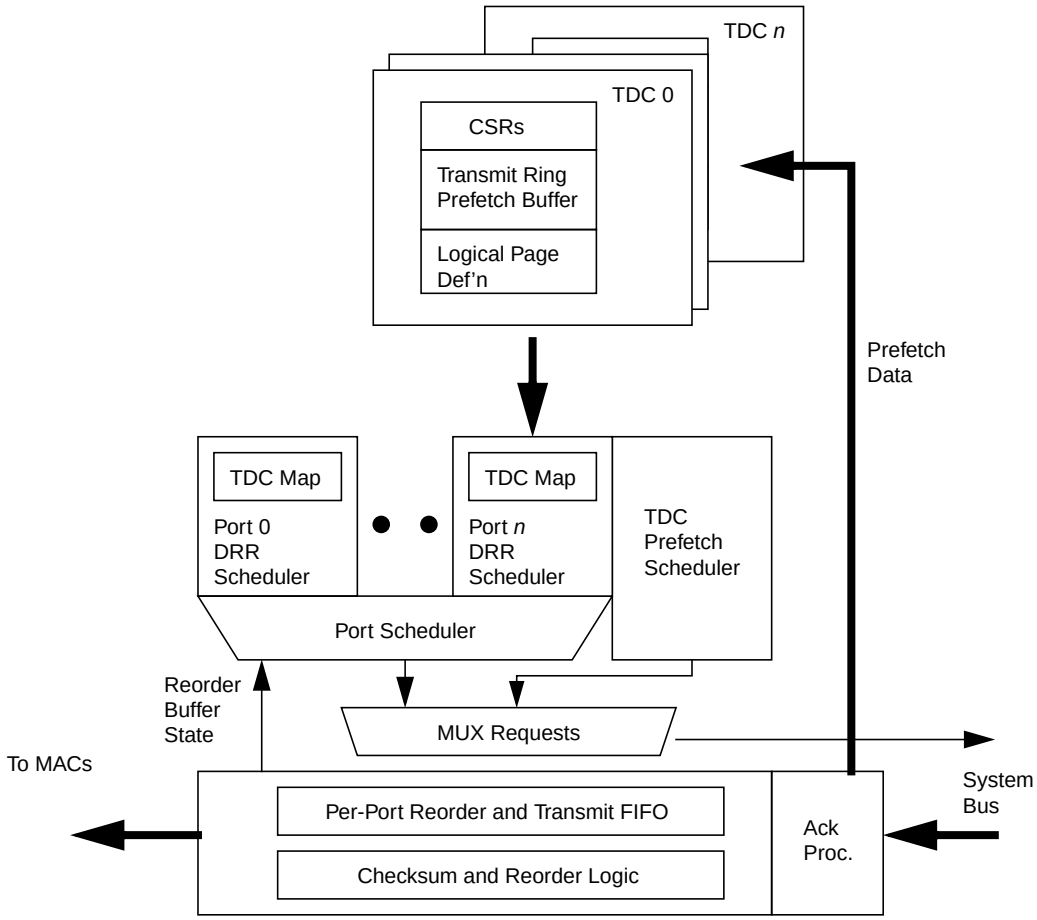

FIGURE 2-5 Transmit Functional Blocks

The transmit data requests and the prefetch requests share the same datapath to memory system. The returned acknowledgment is first processed to decide whether it is a prefetch or a transmit data. Transmit hardware also support checksum offload. This logic is embedded in the Reorder and Transmit FIFO logic.

When the entire packet has been received into the Transmit FIFO, the transmission of the packet is considered to be completed and the state of the DMA channel will be updated through the associated status register. A 12-bit wrap around counter, initialized to 0, keeps track of packets transmitted. Software needs to poll the status registers to determine the status. Alternatively, software may *mark* a packet so that

an interrupt (if enabled) may be issued after the transmission of the packet. Similarly to the receive side, hardware may update the state of the DMA channel to a predefined mailbox after transmitting a marked packet.

Transmit and receive logic fairly share the same memory system interface.

2.2.5 Notes on Register Definition and DMA Addressing

The following convention is used in defining the addresses of one or more register blocks.

Register Name – Short_name (Base) [(count c step s)]

Register Name is the name of the register. Short_name is a short alias of the register. ***Base*** is the base address of block. ***count c*** step s indicates that there are *c* counts of the register in each block, the addresses of the registers are s apart. For example, if Base is $1000\ 0000_{16}$, count is 3, and step is 8, then the addresses for the registers are 10000000, 10000008, and 10000010_{16}. Registers that hardware will not initialize are marked as Xs, and software needs to write proper values to these registers.

Internally NIU is a *little-endian* device. In Neptune, to support 32-bit systems, registers are 32-bit addressable if naturally aligned. All registers are defined to be 64 bits wide, though most of the registers have only 32 bits. Writes to the reserved fields will be ignored and have no side effect. Reads to the reserved fields will be accepted and zeros are returned. Registers have more than 32 bits defined are accessable by software as one PIO. If no exception is specified, side effects are triggered by accessing the most significant 32 bits. Note that not all 64-bit registers need shadowing.

For Neptune, both 64-bit and 32-bit addressing are supported.

CHAPTER 4

Neptune Register Map

4.1 General Information

This chapter describes the various Control and Status Registers(CSRs) mapping in the Network Interface Core Unit that are visible to the software programmer. The PCI Express Configuration Space register map and the PCI Express Interface Module register maps are in their respective chapters. The following abbrevations are used throughout the chapter

R - Read only A register field that is not writeable

RW - Read/Write A register field that is writeable and readable

RAC - Read Auto Clear A register field that is not writeable and clears when read

RW1C - Read/write 1 to clear Writing a 0 to bits in this field has no affect, but writing a 1 to a bit in this field will cause that bit to be set to 0.

4.2 Register Convention

All registers in Neptune/NIU (Network Interface Core Unit) are defined as a 64bit registers.

4.2.1 Register Access Method

In Neptune depending upon the host system, these registers can be accessed in either of the two modes.

- As a 32bit quantity aligned on a 4byte boundary. In thi s case the software needs to do two read or write operations and interpret the entire 64bit quantity as an atomic value. The software should access the lower address first and then the upper address inorder to get consistet results.
- As a 64bit quantity aligned on a 8byte boundary.

4.2.2 Reserved Bits

Some registers contain certain bits that are marked as reserved. These bits should never be set to a value of 1 by software unless explicitly stated. Reads from registers containing reserved bits can return indeterminate values in the reserved bit positions. These reserved bits should be ignored by software.

4.2.3 Reserved and Undefined Address

Any register address not explicitly declared in this specification should be considered to be reserved and should not be written. Such writes will be ignored by the hardware. Reads from reserved or undefined addresses can return indeterminate values unless read values are explicitly stated for specific address.

4.2.4 Initial Values

Most registers define the intial hardware values prior to being programmed. In some cases, the hardware initial values are undefined and are as listed in the text.

4.3 Address Assignment And Multi-Function/Multi-Device Support

Logically, NIU may be viewed as a platform with multiple independent hardware units. In Neptune, NIU is presented by the PCI Express interface as a single device with four functions. PIOs will be addressed with the system allocated PCI address space. The Base Address Registers will be programmed with the allocated values. The PCI Express interface unit will translate the PIO address into the internal address interpreted by NIU hardware. Internal to the NIU PIO block, the following address mapping is supported. Bits [26:24] are used to identify a function and a memory region within a function. There are two types of memory regions: management region and virtualization region. Within a memory region, all hardware registers are visible. In virtualization region, only selected hardware registers are accessible. The latter region is used to support hardware virtualization. The management regions of all functions are simply aliases of each other. Virtualization

regions are different. However, whether certain registers are accessible within a function's management region, and what the access right (either read only or read/write) is, depends on the device function number.

In Neptune, each function has a management region and a virtualization region. These two memory regions are represented by two separate BARs. PIO accesses have to be performed through the address space defined by the BARs, i.e. bits [26:24] are not directly accessable, and software has to access the regions defined by the BARs to access the registers.

Within the NIU address space, all defined registers are 64-bit entities. This does not mean that there is a physical device behind every bit field. In particular, reserved fields in a register are read only, and will return zero's when read. (For example, there may be 32 bit registers in some logic block. These registers are extended to have 64-bit definition, with the higher order 32 bits reserved. On reads, the higher order 32 bits will return 0's.) Write accesses to address space not defined in the PRM will be silently discarded. Read requests to these addresses will cause a 'UR' response to the PCI Express request in Neptune.

In a single partition software model, the drivers associated with different functions may come up independently. The first driver that comes up may attempt to configure the device. The register DEV_FUNC_SR provides a simple test-and-set mechanism and a scratch pad register to facilitate locking and communication (described later in a this section). After configuration, software may decide to make the driver associated with function 0 as the master control. Thus, each internal hardware block has a Function Zero Control (FZC) zone, that normally may be accessed by any function. There is a control register bit, MPC bit in MULTI_PART_CTL register, if set, all the registers within these zones can only be written to via the function zero management space. Registers within this space MAY be read by any function.

The management region of Network Interface Unit occupies a 24 bit address space. Bits [23:20] are used to select internal subsystem within the Network Interface Unit. The following table describes the base address of various subsystem. Most internal blocks have two address regions. These regions are to distinguish between Function-Zero(FZC) and non FZC. Bit 19 of the address bits are set for the region to be in FZC range.

4.4 Network Interface Unit Register Map

The following table contains register map for the Network Interface Core Unit.

TABLE 4-1 Network Interface Unit Register Map

Address Offset	Register Defination	Register Name	Number of Registers and Step Size	Initial Value
PIO+0x10000	Device Function Shared Register	DEV_FUNC_SR		
FZC_PIO+0x00000	Multi Partition Control	MULTI_PART_CTL		
FZC_PIO+0x10000	DMA Channel Binding	DMA_BIND	count 64 step 8	
FZC_PIO+0x20000	Logical Device Group Number	LDG_NUM	count 69 step 8	
PIO_LDSV+0x00000	Logical Device State Vector 0	LDSV0	count 64 step 8192	
PIO_LDSV+0x00008	Logical Device State Vector 1	LDSV1	count 64 step 8192	
PIO_LDSV+0x00010	Logical Device State Vector 2	LDSV2	count 64 step 8192	
PIO_IMASK0+0x00000	Logical Device Interrupt Mask 0	LD_IM0	count 64 step 8192	
PIO_IMASK1+0x00000	Logical Device Interrupt Mask 1	LD_IM1	count 5 step 8192	
PIO_LDSV+0x00018	Logical Device Group Interrupt Management	LDGIMGN	count 64 step 8192	
FZC_PIO+0x00008	Logical Device Group Interrupt Timer Resolution	LDGITMRES		
FZC_PIO+0x10200	System Interrupt Data	SID	count 64 step 8	
FZC_PIO+0x00038	Reset Control	RST_CTL		
FZC_PIO+0x00090	System Error Mask	SYS_ERR_MASK		

TABLE 4-1 Network Interface Unit Register Map

Address Offset	Register Defination	Register Name	Number of Registers and Step Size	Initial Value
FZC_PIO+0x00098	System Error State	SYS_ERR_STAT		
FZC_PIO+0x0010	Dirty TID Control	DIRTY_TID_CTL		
FZC_PIO+0x0018	Dirty TID Status-DIRTY_TID_STAT			
FZC_PIO+0x00060	PIO Debug Select	PIO_DBG_SEL		
FZC_PIO+0x00068	PIO Trainig Vector	PIO_TRAIN_VEC		
FZC_PIO+0x00070	PIO Arbiter Control	PIO_ARB_CTL		
FZC_PIO+0x00078	PIO Arbiter Debug Vector	PIO_ARB_DBG_VEC		
FZC_PIO+0x00020	GPIO Data Out	GPIO_DOUT		
FZC_PIO+0x00028	GPIO Enable	GPIO_EN		
FZC_PIO+0x00030	GPIO Data In	GPIO_DIN		
FZC_PROM+0x040000	EPC PIO Enable	EPC_PIO_EN		
FZC_PROM+0x040008	EPC PIO Status	EPC_PIO_STATUS		
FZC_PROM+0x040020	SPCNCRReg	count 128 step 8		
FZC_PIM+0x0	PIM Control	PIM_CONTROL		
FZC_PIM+0x00008	PIM Debug Training Vector	PIM_DBG_TRAINING_VEC		
FZC_PIM+0x00010	PIM Interrupt Status	PIM_INTR_STATUS		
FZC_PIM+0x00018	PIM Internal Status	PIM_INTERNAL_STATUS		
FZC_PIM+0x00020	PIM Interrupt Mask	PIM_INTR_MASK		
FZC_FFLP+0x00000	Ethernet VLAN Table	ENET_VLAN_TBL	count 4096 step 8	
FZC_FFLP+0x20000	Layer 2 Class	L2_CLS	count 2 step 8	
FZC_FFLP+0x20010	Layer 3 Class	L3_CLS	count 4 step 8	
FZC_FFLP+0x20030	TCAM Key	TCAM_KEY	count 12 step 8	

TABLE 4-1 Network Interface Unit Register Map

Address Offset	Register Defination	Register Name	Number of Registers and Step Size	Initial Value
FZC_FFLP+0x40000	Flow Key	FLOW_KEY	count 12 step 8	
FZC_FFLP+0x20090	TCAM Key 0	TCAM_KEY_0		
FZC_FFLP+0x20098	TCAM Key 1	TCAM_KEY_1		
FZC_FFLP+0x200A0	TCAM Key 2	TCAM_KEY_2		
FZC_FFLP+0x200A8	TCAM Key 3	TCAM_KEY_3		
FZC_FFLP+0x200B0	TCAM Key Mask 0	TCAM_KEY_MASK_0		
FZC_FFLP+0x200B8	TCAM Key Mask 1	TCAM_KEY_MASK_1		
FZC_FFLP+0x200C0	TCAM Key Mask 2	TCAM_KEY_MASK_2		
FZC_FFLP+0x200C8	TCAM Key Mask 3	TCAM_KEY_MASK_3		
FZC_FFLP+0x200D0	TCAM Control	TCAM_CTL		
FZC_FFLP+0x40060	H1 Polynomial	H1POLY		
FZC_FFLP+0x40068	H2 Polynomial	H2POLY		
FZC_FFLP+0x40070	Flow Partition Select	FLW_PRT_SEL	count 8 step 8	
FFLP+0x00000	Hash Table Address	HASH_TBL_ADDR	count 8 step 8192	
FFLP+0x00008	Hash Table Data	HASH_TBL_DATA	count 8 step 8192	
FZC_FFLP+0x20100	FFLP Configuration 1	FFLP_CFG_1		
FZC_FFLP+0x20148	FFLP Debug Training Vector	FFLP_DBG_TRAIN_VCT		
FZC_FFLP+0x20108	TCP Control Flag Mask	TCP_CFLAG_MSK		
FZC_FFLP+0x20110	FCRAM Refresh Timer	FCRAM_REF_TMR		
FZC_FFLP+0x20118	FCRAM Controller Address	FCRAM_FIO_ADDR		
FZC_FFLP+0x20120	FCRAM Controller Data	FCRAM_FIO_DAT		
FZC_FFLP+0x20150	FCRAM PHY Read Latency	FCRAM_PHY_RD_LAT		
FZC_FFLP+0x08000	VLAN Parity Error	FFLP_VLAN_PAR_ERR		
FZC_FFLP+0x200D8	TCAM Error	TCAM_ERR		

TABLE 4-1 Network Interface Unit Register Map

Address Offset	Register Defination	Register Name	Number of Registers and Step Size	Initial Value
FFLP+0x00010	Hash Table Data Error log	HASH_TBL_DATA_LOG	count 8 step 8192	
FZC_FFLP+0x200E0	Hash Table Lookup Error Log 1	HASH_LOOKUP_ERR_LOG1		
FZC_FFLP+0x200E8	Hash Table Lookup Error Log 2	HASH_LOOKUP_ERR_LOG2		
FZC_FFLP+0x20128	FCRAM Error Test 0	FCRAM_ERR_TST0		
FZC_FFLP+0x20130	FCRAM Error Test 1	FCRAM_ERR_TST1		
FZC_FFLP+0x20138	FCRAM Error Test 2	FCRAM_ERR_TST2		
FZC_FFLP+0x20140	FFLP Error Mask	FFLP_ERR_MSK		
FZC_DMC+0x00000	Receive DMA Clock Divider	RX_DMA_CK_DIV		
FZC_DMC+0x00008	Default Port 0 RDC	DEF_PT0_RDC		
FZC_DMC+0x00010	Default Port 1 RDC	DEF_PT1_RDC		
FZC_DMC+0x00018	Default Port 2 RDC	DEF_PT2_RDC		
FZC_DMC+0x00020	Default Port 3 RDC	DEF_PT3_RDC		
FZC_ZCP+0x10000	RDC Table	RDC_TBL	count 128 step 8	
FZC_DMC+0x00070	Receive Addressing Mode	RX_ADDR_MD		
FZC_DMC+0x00028	Port DRR Weight 0	PT_DRR_WT0		
FZC_DMC+0x00030	Port DRR Weight 1	PT_DRR_WT1		
FZC_DMC+0x00038	Port DRR Weight 2	PT_DRR_WT2		
FZC_DMC+0x00040	Port DRR Weight 3	PT_DRR_WT3		
FZC_DMC+0x00048	Port FIFO Usage 0	PT_USE0		
FZC_DMC+0x00050	Port FIFO Usage 1	PT_USE1		
FZC_DMC+0x00058	Port FIFO Usage 2	PT_USE2		
FZC_DMC+0x00060	Port FIFO Usage 3	PT_USE3		
FZC_DMC+0x20000	Receive Logical Page Valid	RX_LOG_PAGE_VLD	count 16 step 0x40	
FZC_DMC+0x20008	Receive Logical Page Mask 1	RX_LOG_MASK1	count 16 step 0x40	

TABLE 4-1 Network Interface Unit Register Map

Address Offset	Register Defination	Register Name	Number of Registers and Step Size	Initial Value
FZC_DMC+0x20010	Receive Logical Page Value 1	RX_LOG_VAL1	count 16 step 0x40	
FZC_DMC+0x20018	Receive Logical Page Mask 2	RX_LOG_MASK2	count 16 step 0x40	
FZC_DMC+0x20020	Receive Logical Page Value 2	RX_LOG_VAL2	count 16 step 0x40	
FZC_DMC+0x20028	Receive Logical Page Relocation 1	RX_LOG_PAGE_RELO 1	count 16 step 0x40	
FZC_DMC+0x20030	Receive Logical Page Relocation 2	RX_LOG_PAGE_RELO 2	count 16 step 0x40	
FZC_DMC+0x20038	Receive Logical Page Handle	RX_LOG_PAGE_HDL	count 16 step 0x40	
FZC_DMC+0x00068	RED Random Number INIT	RED_RAN_INIT		
FZC_DMC+0x30000	RDC RED Parameter 2	RDC_RED_PARA	count 16 step 0x40	
DMC+0x00000	RXDMA Configuration 1	RXDMA_CFIG1	count 16 step 0x200	
DMC+0x00008	RXDMA Configuration 2	RXDMA_CFIG2	count 16 step 0x200	
DMC+0x00010	RBR Configuration A	RBR_CFIG_A	count 16 step 0x200	
DMC+0x00018	RBR Configuration B	RBR_CFIG_B	count 16 step 0x200	
DMC+0x00020	RBR Kick	RBR_KICK	count 16 step 0x200	
DMC+0x00028	RBR Status	RBR_STAT	count 16 step 0x200	
DMC+0x00030	RBR Head High	RBR_HDH	count 16 step 0x200	
DMC+0x00038	RBR Head Low	RBR_HDL	count 16 step 0x200	
DMC+0x00040	RCR Configuration A	RCRCFIG_A	count 16 step 0x200	
DMC+0x00048	RCR Configuration B	RCRCFIG_B	count 16 step 0x200	

TABLE 4-1 Network Interface Unit Register Map

Address Offset	Register Defination	Register Name	Number of Registers and Step Size	Initial Value
DMC+0x00050	RCR Status A	RCRSTAT_A	count 16 step 0x200	
DMC+0x00058	RCR Status B	RCRSTAT_B	count 16 step 0x200	
DMC+0x00060	RCR Status C	RCRSTAT_C	count 16 step 0x200	
DMC+0x00068	Receive DMA ChannelEventMask-	RX_DMA_ENT_MSK	count 16 step 0x200	
DMC+0x00070	Receive DMA Control And Status	RX_DMA_CTL_STAT	count 16 step 0x200	
DMC+0x00098	Receive DMA Control And Status Debug	RX_DMA_CTL_STAT_ DBG	count 16 step 0x200	
DMC+0x00078	RCR Flush	RCR_FLSH	count 16 step 0x200	
DMC+0x00090	Receive Miscellaneous Discard Count	RXMISC	count 16 step 0x200	
FZC_DMC+0x30008	RED Discard Count	RED_DIS_CNT	count 16 step 0x40	
FZC_DMC+0x00078	RDMC Prefetch Parity Error	RDMC_PRE_PAR_ERR		
FZC_DMC+0x00080	RDMC Shadow Parity Error	RDMC_SHA_PAR_ERR		
FZC_DMC+0x00088	RDMC Memory Address	RDMC_MEM_ADDR		
FZC_DMC+0x00090	RDMC Memory Data 0	RDMC_MEM_DAT0		
FZC_DMC+0x00098	RDMC Memory Data 1	RDMC_MEM_DAT1		
FZC_DMC+0x000A0	RDMC Memory Data 2	RDMC_MEM_DAT2		
FZC_DMC+0x000A8	RDMC Memory Data 3	RDMC_MEM_DAT3		
FZC_DMC+0x000B0	RDMC Memory Data 4	RDMC_MEM_DAT4		

TABLE 4-1 Network Interface Unit Register Map

Address Offset	Register Defination	Register Name	Number of Registers and Step Size	Initial Value
FZC_DMC+0x000C0	Receive Control Data FIFO Error Mask	RX_CTL_DAT_FIFO_MASK		
FZC_DMC+0x000B8	Receive Control Data FIFO Error Status	RX_CTL_DAT_FIFO_STAT		
FZC_DMC+0x000D0	Receive Control Data FIFO Error Status Debug	RX_CTL_DAT_FIFO_STAT_DBG		
FZC_DMC+0x000C8	RDMC Training Vector	RDMC_TRAINING_VECTOR		
FZC_IPP	IPP Configuration	IPP_CFIG	count **4** step 0x4000	
FZC_IPP+0x0020	IPP Packet Discard	IPP_PKT_DIS	count **4** step 0x4000	
FZC_IPP+0x0028	IPP Bad Checksum Count	IPP_BAD_CS_CNT	count **4** step 0x4000	
FZC_IPP+0x0030	IPP ECC Error Count	IPP_ECC	count **4** step 0x4000	
FZC_IPP+0x0040	IPP Interrupt Status	IPP_INT_STAT	count **4** step 0x4000	
FZC_IPP+0x0048	IPP Interrupt Mask	IPP_MSK	count **4** step 0x4000	
FZC_IPP+0x0060	IPP PreFIFO Read Data 1	IPP_PFIFO_RD1	count **4** step 0x4000	
FZC_IPP+0x0068	IPP PreFIFO Read Data 2	IPP_PFIFO_RD2	count **4** step 0x4000	
FZC_IPP+0x0070	IPP PreFIFO Read Data 3	IPP_PFIFO_RD3	count **4** step 0x4000	
FZC_IPP+0x0078	IPP PreFIFO Read Data 4	IPP_PFIFO_RD4	count **4** step 0x4000	
FZC_IPP+0x0080	IPP PreFIFO Read Data 5	IPP_PFIFO_RD5	count **4** step 0x4000	
FZC_IPP+0x0088	IPP PreFIFO Write Data 1	IPP_PFIFO_WR1	count **4** step 0x4000	
FZC_IPP+0x0090	IPP PreFIFO Write Data 2	IPP_PFIFO_WR2	count **4** step 0x4000	

TABLE 4-1 Network Interface Unit Register Map

Address Offset	Register Defination	Register Name	Number of Registers and Step Size	Initial Value
FZC_IPP+0x0098	IPP PreFIFO Write Data 3	IPP_PFIFO_WR3	count **4** step 0x4000	
FZC_IPP+0x00A0	IPP PreFIFO Write Data 4	IPP_PFIFO_WR4	count **4** step 0x4000	
FZC_IPP+0x00A8	IPP PreFIFO Write Data 5	IPP_PFIFO_WR5	count **4** step 0x4000	
FZC_IPP+0x00B0	IPP Pre-FIFO Read Pointer	IPP_PFIFO_RD_PTR	count **4** step 0x4000	
FZC_IPP+0x00B8	IPP PreFIFO Write Pointer	IPP_PFIFO_WR_PTR	count **4** step 0x4000	
FZC_IPP+0x00C0	IPP Data FIFO Read Data 1	IPP_DFIFO_RD1	count **4** step 0x4000	
FZC_IPP+0x00C8	IPP Data FIFO Read Data 2	IPP_DFIFO_RD2	count **4** step 0x4000	
FZC_IPP+0x00D0	IPP Data FIFO Read Data 3	IPP_DFIFO_RD3	count **4** step 0x4000	
FZC_IPP+0x00D8	IPP Data FIFO Read Data 4	IPP_DFIFO_RD4	count **4** step 0x4000	
FZC_IPP+0x00E0	IPP Data FIFO Read Data 5	IPP_DFIFO_RD5	count **4** step 0x4000	
FZC_IPP+0x00E8	IPP Data FIFO Write Data 1	IPP_DFIFO_WR1	count **4** step 0x4000	
FZC_IPP+0x00F0	IPP Data FIFO Write Data 2	IPP_DFIFO_WR2	count **4** step 0x4000	
FZC_IPP+0x00F8	IPP Data FIFO Write Data 3	IPP_DFIFO_WR3	count **4** step 0x4000	
FZC_IPP+0x0100	IPP Data FIFO Write Data 4	IPP_DFIFO_WR4	count **4** step 0x4000	
FZC_IPP+0x0108	IPP Data FIFO Write Data 5	IPP_DFIFO_WR5	count **4** step 0x4000	
FZC_IPP+0x0110	IPP Data FIFO Read Pointer	IPP_DFIFO_RD_PTR	count **4** step 0x4000	
FZC_IPP+0x0118	IPP Data FIFO Write Pointer	IPP_DFIFO_WR_PTR	count **4** step 0x4000	
FZC_IPP+0x0120	IPP State Machine	IPP_SM	count **4** step 0x4000	

TABLE 4-1 Network Interface Unit Register Map

Address Offset	Register Defination	Register Name	Number of Registers and Step Size	Initial Value
FZC_IPP+0x0128	IPP Checksum Status	IPP_CS_STAT	count **4** step 0x4000	
FZC_IPP+0x0130	IPP FFLP Checksum Information	IPP_FFLP_CS_INFO	count **4** step 0x4000	
FZC_IPP+0x0138	IPP Debug Select	IPP_DBG_SEL	count **4** step 0x4000	
FZC_IPP+0x0140	IPP Data FIFO ECC Syndrome	IPP_DFIFO_ECC_SYND	count **4** step 0x4000	
FZC_IPP+0x0148	IPP Data FIFO EOP Missed Read Pointer	IPP_DFIFO_EOP_RD_PTR	count **4** step 0x4000	
FZC_IPP+0x0150	IPP ECC Control	IPP_ECC_CTL	count **4** step 0x4000	
FZC_DMC+0x40000	Transmit Logical Page Valid	TX_LOG_PAGE_VLD	count **24** step 0x200	
FZC_DMC+0x40008	Transmit Logical Page Mask 1	TX_LOG_MASK1	count **24** step 0x200	
FZC_DMC+0x40010	Transmit Logical Page Value 1	TX_LOG_VALUE1	count **24** step 0x200	
FZC_DMC+0x40018	Transmit Logical Page Mask 2	TX_LOG_MASK2	count **24** step 0x200	
FZC_DMC+0x40020	Transmit Logical Page Value 2	TX_LOG_VALUE2	count **24** step 0x200	
FZC_DMC+0x40028	Transmit Logical Page Relocation 1	TX_LOG_PAGE_RELO1	count **24** step 0x200	
FZC_DMC+0x40030	Transmit Logical Page Relocation 2	TX_LOG_PAGE_RELO2	count **24** step 0x200	
FZC_DMC+0x40038	Transmit Logical Page Handle	TX_LOG_PAGE_HDL	count **24** step 0x200	
FZC_DMC+0x45000	Transmit Address Mode	TX_ADDR_MD		
DMC+0x40000	Transmit Ring Configuration	TX_RNG_CFIG	count **24** step 0x200	
DMC+0x40010	Transmit Ring Head Low	TX_RING_HDL	count **24** step 0x200	
DMC+0x40018	Transmit Ring Kick	TX_RING_KICK	count **24** step 0x200	

TABLE 4-1 Network Interface Unit Register Map

Address Offset	Register Defination	Register Name	Number of Registers and Step Size	Initial Value
DMC+0x40020	Transmit Event Mask	TX_ENT_MSK	count **24** step 0x200	
DMC+0x40028	Transmit Control And Status	TX_CS	count **24** step 0x200	
DMC + 0x40060	Transmit DMA Interrupt Debug	TDMC_INTR_DBG	count **24** step 0x200	
DMC+0x40068	Transmit Control And Status Debug	TX_CS_DBG	count **24** step 0x200	
DMC+0x40030	TXDMA Mailbox High	TXDMA_MBH	count **24** step 0x200	
DMC+0x40038	TXDMA Mailbox Low	TXDMA_MBL	count **24** step 0x200	
DMC+0x40040	Transmit DMA Prefetch State	TX_DMA_PRE_ST	count **24** step 0x200	
DMC+0x40048	Transmit Ring Error Log High	TX_RNG_ERR_LOGH	count **24** step 0x200	
DMC+0x40050	Transmit Ring Error Log Low	TX_RNG_ERR_LOGL	count **24** step 0x200	
FZC_TXC+0x20028	TXC Port DMA Enable	TXC_PORT_DMA	count **4** step 0x00100	
FZC_TXC+0x00000	TXC DMA Max Burst	TXC_DMA_MAX	count **24** step 0x01000	
FZC_TXC+0x00008	TXC DMA Max Burst Length-	TXC_DMA_MAX_LEN	count **24** step 0x01000	
FZC_DMC+0x45040	TDMC Inject Parity Error	TDMC_INJ_PAR_ERR		
FZC_DMC+0x45080	TDMC Debug Select	TDMC_DBG_SEL		
FZC_DMC+0x45088	TDMC Training Vector	TDMC_TRAINING_VECTOR		
FZC_TXC+0x20000	TXC Control	TXC_CONTROL		
FZC_TXC+0x20008	TXC Training	TXC_TRAINING		
FZC_TXC+0x20010	TXC Debug Select	TXC_DEBUG		
FZC_TXC+0x20018	TXC Maximum Reorder	TXC_MAX_REORDER		

TABLE 4-1 Network Interface Unit Register Map

Address Offset	Register Defination	Register Name	Number of Registers and Step Size	Initial Value
FZC_TXC+0x20020	TXC Port Control	TXC_PORT_CTL	count **4** step 0x00100	
FZC_TXC+0x20030	TXC Packets Stuffed	TXC_PKT_STUFFED	count **4** step 0x00100	
FZC_TXC+0x20038	TXC Packet Transmitted	TXC_PKT_XMIT	count **4** step 0x00100	
FZC_TXC+0x20040	TXC RO ECC Control	TXC_ROECC_CTL	count **4** step 0x00100	
FZC_TXC+0x20048	TXC RO ECC STATE	TXC_ROECC_ST	count **4** step 0x00100	
FZC_TXC+0x20050	TXC RO Data 0	TXC_RO_DATA0	count **4** step 0x00100	
FZC_TXC+0x20058	TXC RO Data 1	TXC_RO_DATA1	count **4** step 0x00100	
FZC_TXC+0x20060	TXC RO Data 2	TXC_RO_DATA2	count **4** step 0x00100	
FZC_TXC+0x20068	TXC RO Data 3	TXC_RO_DATA3	count **4** step 0x00100	
FZC_TXC+0x20070	TXC RO Data 4	TXC_RO_DATA4	count **4** step 0x00100	
FZC_TXC+0x20078	TXC SF ECC Control	TXC_SFECC_CTL	count **4** step 0x00100	
FZC_TXC+0x20080	TXC SF ECC STATE	TXC_SFECC_ST	count **4** step 0x00100	
FZC_TXC+0x20088	TXC SF Data 0	TXC_SF_DATA0	count **4** step 0x00100	
FZC_TXC+0x20090	TXC SF Data 1	TXC_SF_DATA1	count **4** step 0x00100	
FZC_TXC+0x20098	TXC SF Data 2	TXC_SF_DATA2	count **4** step 0x00100	
FZC_TXC+0x200A0	TXC SF Data 3	TXC_SF_DATA3	count **4** step 0x00100	
FZC_TXC+0x200A8	TXC SF Data 4	TXC_SF_DATA4	count **4** step 0x00100	
FZC_TXC+0x200B0	TXC RE-Order TIDs	TXC_RO_TIDS	count **4** step 0x00100	

TABLE 4-1 Network Interface Unit Register Map

Address Offset	Register Defination	Register Name	Number of Registers and Step Size	Initial Value
FZC_TXC+0x200B8	TXC RO STATE0	TXC_RO_STATE0	count **4** step0x00100	
FZC_TXC+0x200C0	TXC RO STATE1	TXC_RO_STATE1	count **4** step0x00100	
FZC_TXC+0x200C8	TXC RO STATE2	TXC_RO_STATE2	count **4** step 0x00100	
FZC_TXC+0x200D0	TXC RO STATE3	TXC_RO_STATE3	count **4** step 0x00100	
FZC_TXC+0x200D8	TXC RO_ST_Control	TXC_RO_CTL	count **4** step 0x00100	
FZC_TXC+0x200E0	TXC RO_ST_Data0	TXC_RO_ST_DATA0	count **4** step 0x00100	
FZC_TXC+0x200E8	TXC RO_ST_Data1	TXC_RO_ST_DATA1	count **4** step 0x00100	
FZC_TXC+0x200F0	TXC RO_ST_Data2	TXC_RO_ST_DATA2	count **4** step 0x00100	
FZC_TXC+0x200F8	TXC RO_ST_Data3	TXC_RO_ST_DATA3	count **4** step 0x00100	
FZC_TXC+0x20100	Port Packets Request	TXC_PORT_PACKET_REQ	count **4** step 0x00100	
FZC_TXC+0x20420	TXC Interrupt_Stat_Debug	TXC_INT_STAT_DBG		
FZC_TXC+0x20428	TXC Interrupt_Stat	TXC_INT_STAT		
FZC_TXC+0x20430	TXC Inerrupt_Mask	TXC_INT_MASK		
FZC_MAC+0x00000	TxMAC Software Reset Command	XTxMAC_SW_RST	count 2 step 0x06000	
FZC_MAC+0x00008	RxMAC Software Reset Command	XRxMAC_SW_RST	count 2 step 0x06000	
FZC_MAC+0x00020	Tx_xMAC Status	XTxMAC_STATUS	count 2 step 0x06000	
FZC_MAC+0x00028	Rx_xMAC Status	XRxMAC_STATUS	count 2 step 0x06000	
FZC_MAC+0x00030	xMAC Flow Control Status	XMAC_FC_STAT	count 2 step 0x06000	

TABLE 4-1 Network Interface Unit Register Map

Address Offset	Register Defination	Register Name	Number of Registers and Step Size	Initial Value
FZC_MAC+0x00040	Tx_xMAC Mask	XTxMAC_STAT_MSK	count 2 step 0x06000	
FZC_MAC+0x00048	Rx_xMAC Mask	XRxMAC_STAT_MSK	count 2 step 0x06000	
FZC_MAC+0x00050	xMAC Flow Control Mask	XMAC_FC_MSK	count 2 step 0x06000	
FZC_MAC+0x00060	xMAC Configuration	XMAC_CONFIG	count 2 step 0x06000	
FZC_MAC+0x00080	xMAC Inter-Packet Gap	XMAC_IPG	count 2 step 0x06000	
FZC_MAC+0x00088	xMAC Minimum Frame Size	XMAC_MIN	count 2 step 0x06000	
FZC_MAC+0x00090	xMAC Maximum Frame Size	XMAC_MAX	count 2 step 0x06000	
FZC_MAC+0x00100	xMAC Receive Byte Counter	RxMAC_BT_CNT	count 2 step 0x06000	
FZC_MAC+0x00000	xMAC Receive Broad-Cast Frame Counter	RxMAC_BC_FRM_CNT	count 2 step 0x0108	
FZC_MAC+0x00110	xMAC Receive Multi-Cast Frame Counter	RxMAC_MC_FRM_CNT	count 2 step 0x06000	
FZC_MAC+0x00118	xMAC Receive Fragments Counter	RxMAC_FRAG_CNT	count 2 step 0x06000	
FZC_MAC+0x00120	xMAC Receive 64B Frame Counter	RxMAC_HIST_CNT1	count 2 step 0x06000	
FZC_MAC+0x00128	xMAC Receive 64B-127B Frame Counter	RxMAC_HIST_CNT2	count 2 step 0x06000	
FZC_MAC+0x00130	xMAC Receive 128B-255B Frame Counter	RxMAC_HIST_CNT3	count 2 step 0x06000	
FZC_MAC+0x00138	xMAC Receive 256B-511B Frame Counter	RxMAC_HIST_CNT4	count 2 step 0x06000	
FZC_MAC+0x00140	xMAC Receive 256B-511B Frame Counter	RxMAC_HIST_CNT4	count 2 step 0x06000	
FZC_MAC+0x00148	xMAC Receive 1024B-1522B Frame Counter	RxMAC_HIST_CNT6	count 2 step 0x06000	

TABLE 4-1 Network Interface Unit Register Map

Address Offset	Register Defination	Register Name	Number of Registers and Step Size	Initial Value
FZC_MAC+0x00188	xMAC Receive Jumbo Frame Counter	RxMAC_HIST_CNT7	count 2 step 0x06000	
FZC_MAC+0x00150	xMAC Receive Max Packet Length Error Counter	RxMAC_MPSZER_CNT	count 2 step 0x06000	
FZC_MAC+0x00158	xMAC Receive CRC Error Counter	RxMAC_CRC_ER_CNT	count 2 step 0x06000	
FZC_MAC+0x00160	xMAC Receive Code Violation Error Counter	RxMAC_CD_VIO_CNT	count 2 step 0x06000	
FZC_MAC+0x00170	xMAC Transmit Frame Counter	TxMAC_FRM_CNT	count 2 step 0x06000	
FZC_MAC+0x00178	xMAC Transmit Byte Counter	TxMAC_BYTE_CNT	count 2 step 0x06000	
FZC_MAC+0x00180	xMAC Link Fault Counter	LINK_FAULT_CNT	count 2 step 0x06000	
FZC_MAC+0x001A8	xMAC State Machines	XMAC_SM_REG	count 2 step 0x06000	
FZC_MAC+0x001B0	xMAC Internal Signals 1	XMAC_INTERN1	count 2 step 0x06000	
FZC_MAC+0x001B8	xMAC Internal Signals 2	XMAC_INTERN2	count 2 step 0x06000	
FZC_MAC+0x000A0	xMAC Address 0	XMAC_ADDR0	count 2 step 0x06000	
FZC_MAC+0x000A8	xMAC Address 1	XMAC_ADDR1	count 2 step 0x06000	
FZC_MAC+0x000B0	xMAC Address 2	XMAC_ADDR2	count 2 step 0x06000	
FZC_MAC+0x00208	xMAC Alternate Address Compare Enable	XMAC_ADDR_CMPEN	count 2 step 0x06000	
FZC_MAC+0x00218	xMAC Alternate Address 0 LSB	XMAC_ADDR3	count 2 step 0x06000	
FZC_MAC+0x00220	xMAC Alternate Address 0 MID	XMAC_ADDR4	count 2 step 0x06000	

TABLE 4-1 Network Interface Unit Register Map

Address Offset	Register Defination	Register Name	Number of Registers and Step Size	Initial Value
FZC_MAC+0x00228	xMAC Alternate Address 0 MSB	XMAC_ADDR5	count 2 step 0x06000	
FZC_MAC+0x00230	xMAC Alternate Address 1 LSB	XMAC_ADDR6	count 2 step 0x06000	
FZC_MAC+0x00238	xMAC Alternate Address 1 MID	XMAC_ADDR7	count 2 step 0x06000	
FZC_MAC+0x00240	xMAC Alternate Address 1 MSB	XMAC_ADDR8	count 2 step 0x06000	
FZC_MAC+0x00248	xMAC Alternate Address 2 LSB	XMAC_ADDR9	count 2 step 0x06000	
FZC_MAC+0x00250	xMAC Alternate Address 2 MID	XMAC_ADDR10	count 2 step 0x06000	
FZC_MAC+0x00258	xMAC Alternate Address 2 MSB	XMAC_ADDR11	count 2 step 0x06000	
FZC_MAC+0x00260	xMAC Alternate Address 3 LSB	XMAC_ADDR12	count 2 step 0x06000	
FZC_MAC+0x00268	xMAC Alternate Address 3 MID	XMAC_ADDR13	count 2 step 0x06000	
FZC_MAC+0x00270	xMAC Alternate Address 3 MSB	XMAC_ADDR14	count 2 step 0x06000	
FZC_MAC+0x00278	xMAC Alternate Address 4 LSB	XMAC_ADDR15	count 2 step 0x06000	
FZC_MAC+0x00280	xMAC Alternate Address 4 MID	XMAC_ADDR16	count 2 step 0x06000	
FZC_MAC+0x00288	xMAC Alternate Address 4 MSB	XMAC_ADDR17	count 2 step 0x06000	
FZC_MAC+0x00290	xMAC Alternate Address 5 LSB	XMAC_ADDR18	count 2 step 0x06000	
FZC_MAC+0x00298	xMAC Alternate Address 5 MID	XMAC_ADDR19	count 2 step 0x06000	
FZC_MAC+0x002A0	xMAC Alternate Address 5 MSB	XMAC_ADDR20	count 2 step 0x06000	
FZC_MAC+0x002A8	xMAC Alternate Address 6 LSB	XMAC_ADDR21	count 2 step 0x06000	
FZC_MAC+0x002B0	xMAC Alternate Address 6 MID	XMAC_ADDR22	count 2 step 0x06000	

TABLE 4-1 Network Interface Unit Register Map

Address Offset	Register Defination	Register Name	Number of Registers and Step Size	Initial Value
FZC_MAC+0x002B8	xMAC Alternate Address 6 MSB	XMAC_ADDR23	count 2 step 0x06000	
FZC_MAC+0x002C0	xMAC Alternate Address 7 LSB	XMAC_ADDR24	count 2 step 0x06000	
FZC_MAC+0x002C8	xMAC Alternate Address 7 MID	XMAC_ADDR25	count 2 step 0x06000	
FZC_MAC+0x002D0	xMAC Alternate Address 7 MSB	XMAC_ADDR26	count 2 step 0x06000	
FZC_MAC+0x002D8	xMAC Alternate Address 8 LSB	XMAC_ADDR27	count 2 step 0x06000	
FZC_MAC+0x002E0	xMAC Alternate Address 8 MID	XMAC_ADDR28	count 2 step 0x06000	
FZC_MAC+0x002E8	xMAC Alternate Address 8 MSB	XMAC_ADDR29	count 2 step 0x06000	
FZC_MAC+0x002F0	xMAC Alternate Address 9 LSB	XMAC_ADDR30	count 2 step 0x06000	
FZC_MAC+0x002F8	xMAC Alternate Address 9 MID	XMAC_ADDR31	count 2 step 0x06000	
FZC_MAC+0x00300	xMAC Alternate Address 9 MSB	XMAC_ADDR32	count 2 step 0x06000	
FZC_MAC+0x00308	xMAC Alternate Address 10 LSB	XMAC_ADDR33	count 2 step 0x06000	
FZC_MAC+0x00310	xMAC Alternate Address 10 MID	XMAC_ADDR34	count 2 step 0x06000	
FZC_MAC+0x00318	xMAC Alternate Address 10 MSB	XMAC_ADDR35	count 2 step 0x06000	
FZC_MAC+0x00320	xMAC Alternate Address 11 LSB	XMAC_ADDR36	count 2 step 0x06000	
FZC_MAC+0x00328	xMAC Alternate Address 11 MID	XMAC_ADDR37	count 2 step 0x06000	
FZC_MAC+0x00330	xMAC Alternate Address 11 MSB	XMAC_ADDR38	count 2 step 0x06000	
FZC_MAC+0x00338	xMAC Alternate Address 12 LSB	XMAC_ADDR39	count 2 step 0x06000	
FZC_MAC+0x00340	xMAC Alternate Address 12 MID	XMAC_ADDR40	count 2 step 0x06000	

TABLE 4-1 Network Interface Unit Register Map

Address Offset	Register Defination	Register Name	Number of Registers and Step Size	Initial Value
FZC_MAC+0x00348	xMAC Alternate Address 12 MSB	XMAC_ADDR41	count 2 step 0x06000	
FZC_MAC+0x00350	xMAC Alternate Address 13 LSB	XMAC_ADDR42	count 2 step 0x06000	
FZC_MAC+0x00358	xMAC Alternate Address 13 MID	XMAC_ADDR43	count 2 step 0x06000	
FZC_MAC+0x00360	xMAC Alternate Address 13 MSB	XMAC_ADDR44	count 2 step 0x06000	
FZC_MAC+0x00368	xMAC Alternate Address 14 LSB	XMAC_ADDR45	count 2 step 0x06000	
FZC_MAC+0x00370	xMAC Alternate Address 14 MID	XMAC_ADDR46	count 2 step 0x06000	
FZC_MAC+0x00378	xMAC Alternate Address 14 MSB	XMAC_ADDR47	count 2 step 0x06000	
FZC_MAC+0x00380	xMAC Alternate Address 15 LSB	XMAC_ADDR48	count 2 step 0x06000	
FZC_MAC+0x00388	xMAC Alternate Address 15 MID	XMAC_ADDR49	count 2 step 0x06000	
FZC_MAC+0x0390	xMAC Alternate Address 15 MSB	XMAC_ADDR50	count 2 step 0x06000	
FZC_MAC+0x00818	xMAC Address Filter LSB	XMAC_ADD_FILT0	count 2 step 0x06000	
FZC_MAC+0x00820	xMAC Address Filter MID	XMAC_ADD_FILT1	count 2 step 0x06000	
FZC_MAC+0x00828	xMAC Address Filter MSB	XMAC_ADD_FILT2	count 2 step 0x06000	
FZC_MAC+0x00830	xMAC Address Filter Mask MSB	XMAC_ADD_FILT12_ MASK	count 2 step 0x06000	
FZC_MAC+0x00838	xMAC Address Filter Mask LSB	XMAC_ADD_FILT00_ MASK	count 2 step 0x06000	
FZC_MAC+0x00840	xMAC Hash Table 0	XMAC_HASH_TBL0	count 2 step 0x06000	
FZC_MAC+0x00848	xMAC Hash Table 1	XMAC_HASH_TBL1	count 2 step 0x06000	
FZC_MAC+0x00850	xMAC Hash Table 2	XMAC_HASH_TBL2	count 2 step 0x06000	

TABLE 4-1 Network Interface Unit Register Map

Address Offset	Register Defination	Register Name	Number of Registers and Step Size	Initial Value
FZC_MAC+0x00858	xMAC Hash Table 3	XMAC_HASH_TBL3	count 2 step 0x06000	
FZC_MAC+0x00860	xMAC Hash Table 4	XMAC_HASH_TBL4	count 2 step 0x06000	
FZC_MAC+0x00868	xMAC Hash Table 5	XMAC_HASH_TBL5	count 2 step 0x06000	
FZC_MAC+0x00870	xMAC Hash Table 6	XMAC_HASH_TBL6	count 2 step 0x06000	
FZC_MAC+0x00878	xMAC Hash Table 7	XMAC_HASH_TBL7	count 2 step 0x06000	
FZC_MAC+0x00880	xMAC Hash Table 8	XMAC_HASH_TBL8	count 2 step 0x06000	
FZC_MAC+0x00888	xMAC Hash Table 9	XMAC_HASH_TBL9	count 2 step 0x06000	
FZC_MAC+0x00890	xMAC Hash Table 10	XMAC_HASH_TBL10	count 2 step 0x06000	
FZC_MAC+0x00898	xMAC Hash Table 11	XMAC_HASH_TBL11	count 2 step 0x06000	
FZC_MAC+0x008A0	xMAC Hash Table 12	XMAC_HASH_TBL12	count 2 step 0x06000	
FZC_MAC+0x008A8	xMAC Hash Table 13	XMAC_HASH_TBL13	count 2 step 0x06000	
FZC_MAC+0x008B0	xMAC Hash Table 14	XMAC_HASH_TBL14	count 2 step 0x06000	
FZC_MAC+0x008B8	xMAC Hash Table 15	XMAC_HASH_TBL15	count 2 step 0x06000	
FZC_MAC+0x00900	xMAC Host Info 0	XMAC_HOST_INFO0	count 2 step 0x06000	
FZC_MAC+0x00908	xMAC Host Info 1	XMAC_HOST_INFO1	count 2 step 0x06000	
FZC_MAC+0x00910	xMAC Host Info 2	XMAC_HOST_INFO2	count 2 step 0x06000	
FZC_MAC+0x00900	xMAC Host Info 0	XMAC_HOST_INFO0	count 2 step 0x06000	
FZC_MAC+0x00918	xMAC Host Info 3	XMAC_HOST_INFO3	count 2 step 0x06000	

TABLE 4-1 Network Interface Unit Register Map

Address Offset	Register Defination	Register Name	Number of Registers and Step Size	Initial Value
FZC_MAC+0x00920	xMAC Host Info 4	XMAC_HOST_INFO4	count 2 step 0x06000	
FZC_MAC+0x00928	xMAC Host Info 5	XMAC_HOST_INFO5	count 2 step 0x06000	
FZC_MAC+0x00930	xMAC Host Info 6	XMAC_HOST_INFO6	count 2 step 0x06000	
FZC_MAC+0x00938	xMAC Host Info 7	XMAC_HOST_INFO7	count 2 step 0x06000	
FZC_MAC+0x00940	xMAC Host Info 8	XMAC_HOST_INFO8	count 2 step 0x06000	
FZC_MAC+0x00948	xMAC Host Info 9	XMAC_HOST_INFO9	count 2 step 0x06000	
FZC_MAC+0x00950	xMAC Host Info 10	XMAC_HOST_INFO10	count 2 step 0x06000	
FZC_MAC+0x00958	xMAC Host Info 11	XMAC_HOST_INFO11	count 2 step 0x06000	
FZC_MAC+0x00960	xMAC Host Info 12	XMAC_HOST_INFO12	count 2 step 0x06000	
FZC_MAC+0x00968	xMAC Host Info 13	XMAC_HOST_INFO13	count 2 step 0x06000	
FZC_MAC+0x00970	xMAC Host Info 14	XMAC_HOST_INFO14	count 2 step 0x06000	
FZC_MAC+0x00978	xMAC Host Info 15	XMAC_HOST_INFO15	count 2 step 0x06000	
FZC_MAC+0x00980	xMAC Host Info 16	XMAC_HOST_INFO16	count 2 step 0x06000	
FZC_MAC+0x00988	xMAC Host Info 17	XMAC_HOST_INFO17	count 2 step 0x06000	
FZC_MAC+0x00990	xMAC Host Info 18	XMAC_HOST_INFO18	count 2 step 0x06000	
FZC_MAC+0x00998	xMAC Host Info 19	XMAC_HOST_INFO19	count 2 step 0x06000	
FZC_MAC+0x00B80	xMAC Preamble Data 0	XMAC_PA_DATA0	count 2 step 0x06000	
FZC_MAC+0x00B88	xMAC Preamble Data 1	XMAC_PA_DATA1	count 2 step 0x06000	

TABLE 4-1 Network Interface Unit Register Map

Address Offset	Register Defination	Register Name	Number of Registers and Step Size	Initial Value
FZC_MAC+0x00B90	xMAC Debug Select	XMAC_DEBUG_SEL	count 2 step 0x06000	
FZC_MAC+0x00B98	xMAC Training Vector	XMAC_TRAIN_VEC	count 2 step 0x06000	
FZC_MAC+0x0C000	BTxMAC Software Reset	BTxMAC_SW_RST	count 2 step 0x04000	
FZC_MAC+0x0C008	BRxMAC Software Reset	BRxMAC_SW_RST	count 2 step 0x04000	
FZC_MAC+0x0C010	BMAC Send Pause	MAC_SEND_PAUSE	count 2 step 0x04000	
FZC_MAC+0x0C020	BTxMAC Status	BTxMAC_STATUS	count 2 step 0x04000	
FZC_MAC+0x0C028	BRxMAC Status	BRxMAC_STATUS	count 2 step 0x04000	
FZC_MAC+0x0C030	BMAC Flow Control Status	BMAC_FC_STAT	count 2 step 0x04000	
FZC_MAC+0x0C040	BTxMAC Mask	BTxMAC_STAT_MSK	count 2 step 0x04000	
FZC_MAC+0x0C048	BRxMAC Mask	BRxMAC_STAT_MSK	count 2 step 0x04000	
FZC_MAC+0x0C050	BMAC Flow Control Mask	BMAC_FC_MSK	count 2 step 0x04000	
FZC_MAC+0x0C060	TxMAC Configuration	TxMAC_CONFIG	count 2 step 0x04000	
FZC_MAC+0x0C068	RxMAC Configuration	RxMAC_CONFIG	count 2 step 0x04000	
FZC_MAC+0x0C070	MAC Control Configuration	MAC_CTRL_CONFIG	count 2 step 0x04000	
FZC_MAC+0x0C078	MAC XIF Configuration	MAC_XIF_CONFIG	count 2 step 0x04000	
FZC_MAC+0x0C0A0	MAC Minimum Frame Size	BMAC_MIN	count 2 step 0x04000	
FZC_MAC+0x0C0A8	MAC Maximum Frame Size	BMAC_MAX	count 2 step 0x04000	
FZC_MAC+0x0C0B0	MAC Preamble Size	MAC_PA_SIZE	count 2 step 0x04000	

TABLE 4-1 Network Interface Unit Register Map

Address Offset	Register Defination	Register Name	Number of Registers and Step Size	Initial Value
FZC_MAC+0x0C0C8	MAC Control Type	MAC_CTRL_TYPE	count 2 step 0x04000	
FZC_MAC+0x0C100	MAC Unique Address 0	BMAC_ADDR0	count 2 step 0x04000	
FZC_MAC+0x0C108	MAC Unique Address 1	BMAC_ADDR1	count 2 step 0x04000	
FZC_MAC+0x0C110	MAC Unique Address 2	BMAC_ADDR2	count 2 step 0x04000	
FZC_MAC+0x0C118	MAC Alternate Address 0_0	MAC_ADDR3	count 2 step 0x04000	
FZC_MAC+0x0C120	MAC Alternate Address 0_1	MAC_ADDR4	count 2 step 0x04000	
FZC_MAC+0x0C128	MAC Alternate Address 0_2	MAC_ADDR5	count 2 step 0x04000	
FZC_MAC+0x0C130	MAC Alternate Address 1_0	MAC_ADDR6	count 2 step 0x04000	
FZC_MAC+0x0C138	MAC Alternate Address 1_1	MAC_ADDR7	count 2 step 0x04000	
FZC_MAC+0x0C140	MAC Alternate Address 1_2	MAC_ADDR8	count 2 step 0x04000	
FZC_MAC+0x0C148	MAC Alternate Address 2_0	MAC_ADDR9	count 2 step 0x04000	
FZC_MAC+0x0C150	MAC Alternate Address 2_1	MAC_ADDR10	count 2 step 0x04000	
FZC_MAC+0x0C158	MAC Alternate Address 2_2	MAC_ADDR11	count 2 step 0x04000	
FZC_MAC+0x0C160	MAC Alternate Address 3_0	MAC_ADDR12	count 2 step 0x04000	
FZC_MAC+0x0C168	MAC Alternate Address 3_1	MAC_ADDR13	count 2 step 0x04000	
FZC_MAC+0x0C170	MAC Alternate Address 3_2	MAC_ADDR14	count 2 step 0x04000	
FZC_MAC+0x0C178	MAC Alternate Address 4_0	MAC_ADDR15	count 2 step 0x04000	
FZC_MAC+0x0C180	MAC Alternate Address 4_1	MAC_ADDR16	count 2 step 0x04000	

TABLE 4-1 Network Interface Unit Register Map

Address Offset	Register Defination	Register Name	Number of Registers and Step Size	Initial Value
FZC_MAC+0x0C188	MAC Alternate Address 4_2	MAC_ADDR17	count 2 step 0x04000	
FZC_MAC+0x0C190	MAC Alternate Address 5_0	MAC_ADDR18	count 2 step 0x04000	
FZC_MAC+0x0C198	MAC Alternate Address 5_1	MAC_ADDR19	count 2 step 0x04000	
FZC_MAC+0x0C1A0	MAC Alternate Address 5_2	MAC_ADDR20	count 2 step 0x04000	
FZC_MAC+0x0C1A8	MAC Alternate Address 6_0	MAC_ADDR21	count 2 step 0x04000	
FZC_MAC+0x0C1B0	MAC Alternate Address 6_1	MAC_ADDR22	count 2 step 0x04000	
FZC_MAC+0x0C1B8	MAC Alternate Address 6_2	MAC_ADDR23	count 2 step 0x04000	
FZC_MAC+0x0C268	MAC Flow Control Address 0	MAC_FC_ADDR0	count 2 step 0x04000	
FZC_MAC+0x0C270	MAC Flow Control Address 1	MAC_FC_ADDR1	count 2 step 0x04000	
FZC_MAC+0x0C278	MAC Flow Control Address 2	MAC_FC_ADDR2	count 2 step 0x04000	
FZC_MAC+0x0C298	MAC Address Filter LSB	MAC_ADD_FILT0	count 2 step 0x04000	
FZC_MAC+0x0C2A0	MAC Address Filter MID	MAC_ADD_FILT1	count 2 step 0x04000	
FZC_MAC+0x0C2A8	MAC Address Filter MSB	MAC_ADD_FILT2	count 2 step 0x04000	
FZC_MAC+0x0C2B0	MAC Address Filter Mask MSB	MAC_ADD_FILT12_MASK	count 2 step 0x04000	
FZC_MAC+0x0C2B8	MAC Address Filter Mask LSB	MAC_ADD_FILT00_MASK	count 2 step 0x04000	
FZC_MAC+0x0C2C0	MAC Hash Table 0	MAC_HASH_TBL0	count 2 step 0x04000	
FZC_MAC+0x0C2C8	MAC Hash Table 1	MAC_HASH_TBL1	count 2 step 0x04000	
FZC_MAC+0x0C2D0	MAC Hash Table 2	MAC_HASH_TBL2	count 2 step 0x04000	

TABLE 4-1 Network Interface Unit Register Map

Address Offset	Register Defination	Register Name	Number of Registers and Step Size	Initial Value
FZC_MAC+0x0C2D8	MAC Hash Table 3	MAC_HASH_TBL3	count 2 step 0x04000	
FZC_MAC+0x0C2E0	MAC Hash Table 4	MAC_HASH_TBL4	count 2 step 0x04000	
FZC_MAC+0x0C2E8	MAC Hash Table 5	MAC_HASH_TBL5	count 2 step 0x04000	
FZC_MAC+0x0C2F0	MAC Hash Table 6	MAC_HASH_TBL6	count 2 step 0x04000	
FZC_MAC+0x0C2F8	MAC Hash Table 7	MAC_HASH_TBL7	count 2 step 0x04000	
FZC_MAC+0x0C300	MAC Hash Table 8	MAC_HASH_TBL8	count 2 step 0x04000	
FZC_MAC+0x0C308	MAC Hash Table 9	MAC_HASH_TBL9	count 2 step 0x04000	
FZC_MAC+0x0C310	MAC Hash Table 10	MAC_HASH_TBL10	count 2 step 0x04000	
FZC_MAC+0x0C318	MAC Hash Table 11	MAC_HASH_TBL11	count 2 step 0x04000	
FZC_MAC+0x0C320	MAC Hash Table 12	MAC_HASH_TBL12	count 2 step 0x04000	
FZC_MAC+0x0C328	MAC Hash Table 13	MAC_HASH_TBL13	count 2 step 0x04000	
FZC_MAC+0x0C330	MAC Hash Table 14	MAC_HASH_TBL14	count 2 step 0x04000	
FZC_MAC+0x0C338	MAC Hash Table 15	MAC_HASH_TBL15	count 2 step 0x04000	
FZC_MAC+0x0C370	BMAC Receive Frame Counter	RxMAC_FRM_CNT	count 2 step 0x04000	
FZC_MAC+0x0C378	BMAC Receive Length Error Counter	MAC_LEN_ER_CNT	count 2 step 0x04000	
FZC_MAC+0x0C380	BMAC Receive Alignment Error Counter	BMAC_AL_ER_CNT	count 2 step 0x04000	
FZC_MAC+0x0C388	BMAC Receive CRC Error Counter	BMAC_CRCL_ER_CNT	count 2 step 0x04000	

TABLE 4-1 Network Interface Unit Register Map

Address Offset	Register Defination	Register Name	Number of Registers and Step Size	Initial Value
FZC_MAC+0x0C390	BMAC Receive Code Violation Counter	BMAC_CD_VIO_CNT	count 2 step 0x04000	
FZC_MAC+0x0C3A0	BMAC State Machines	BMAC_SM_REG	count 2 step 0x04000	
FZC_MAC+0x0C3F8	BMAC Alternate Address Compare Enables	BMAC_ALTAD_CMPEN	count 2 step 0x04000	
FZC_MAC+0x0C400	bMAC Host Info 0	bMAC_HOST_INFO0	count 2 step 0x04000	
FZC_MAC+0x0C408	bMAC Host Info 1	bMAC_HOST_INFO1	count 2 step 0x04000	
FZC_MAC+0x0C410	bMAC Host Info 2	bMAC_HOST_INFO2	count 2 step 0x04000	
FZC_MAC+0x0C418	bMAC Host Info 3	bMAC_HOST_INFO3	count 2 step 0x04000	
FZC_MAC+0x0C420	bMAC Host Info 4	bMAC_HOST_INFO4	count 2 step 0x04000	
FZC_MAC+0x0C428	bMAC Host Info 5	bMAC_HOST_INFO5	count 2 step 0x04000	
FZC_MAC+0x0C430	bMAC Host Info 6	bMAC_HOST_INFO6	count 2 step 0x04000	
FZC_MAC+0x0C438	bMAC Host Info 7	bMAC_HOST_INFO7	count 2 step 0x04000	
FZC_MAC+0x0C440	bMAC Host Info 8	bMAC_HOST_INFO8	count 2 step 0x04000	
FZC_MAC+0x0C448	BMAC Transmit Byte Counter	BTxMAC_BYTE_CNT_DEFAULT	count 2 step 0x04000	
FZC_MAC+0x0C450	BMAC Transmit Frame Counter	BTxMAC_FRM_CNT_DEFAULT	count 2 step 0x04000	
FZC_MAC+0x0C458	BMAC Receive Byte Counter	BRxMAC_BYTE_CNT_DEFAULT	count 2 step 0x04000	
FZC_MAC+0x02000	XPCS Control 1	BASE10G_CONTROL1	count 2 step 0x06000	
FZC_MAC+0x02008	XPCS Status 1	BASE10G_STATUS1	count 2 step 0x06000	

TABLE 4-1 Network Interface Unit Register Map

Address Offset	Register Defination	Register Name	Number of Registers and Step Size	Initial Value
FZC_MAC+0x02010	XPCS Device Identifier	BASE10G_DEVICE_ID ENTIFIER	count 2 step 0x06000	
FZC_MAC+0x02018	XPCS Speed Ability	BASE10G_SPEED_ABIL ITY	count 2 step 0x06000	
FZC_MAC+0x02020	XPCS On-Chip Devices	BASE10G_DEVICES_IN _PACKAGE	count 2 step 0x06000	
FZC_MAC+0x02028	XPCS Control 2	BASE10G_CONTROL2	count 2 step 0x06000	
FZC_MAC+0x02030	XPCS Status 2	BASE10G_STATUS2	count 2 step 0x06000	
FZC_MAC+0x02038	XPCS Package Identifier	BASE10G_PACKAGE_I DENTIFIER	count 2 step 0x06000	
FZC_MAC+0x02040	XPCS Status	BASE10G_STATUS	count 2 step 0x06000	
FZC_MAC+0x02048	XPCS Jitter Test Control	BASE10G_TEST_CONT ROL	count 2 step 0x06000	
FZC_MAC+0x02050	XPCS Test Config.	BASE10G_CFG_VEND OR1	count 2 step 0x06000	
FZC_MAC+0x02058	XPCS Diag.	BASE10G_DIAGNOSTI C_VENDOR2	count 2 step 0x06000	
FZC_MAC+0x02060	XPCS Mask 1	BASE10G_MASK1	count 2 step 0x06000	
FZC_MAC+0x02068	XPCS Packet Counter	BASE10G_PACKET_CO UNTER	count 2 step 0x06000	
FZC_MAC+0x02070	XPCS Transmit State Machine	BASE10G_TX_STATE_ MACHINE	count 2 step 0x06000	
FZC_MAC+0x02078	XPCS Deskew Error Counter	BASE10G_DESKEW_E RROR_COUNTER	count 2 step 0x06000	
FZC_MAC+0x02080	XPCS Symbol Error Counter 01	BASE10G_SYMBOL_ER ROR_COUNTER_01	count 2 step 0x06000	
FZC_MAC+0x02088	XPCS Symbol Error Counter 23	BASE10G_SYMBOL_ER ROR_COUNTER_23	count 2 step 0x06000	
FZC_MAC+0x02090	XPCS Training Vector	BASE10G_TRAIN_VEC TOR	count 2 step 0x06000	
FZC_MAC+0x04000	PCS MII Control	PCS_MII_CTL	count 2 step 0x06000	

TABLE 4-1 Network Interface Unit Register Map

Address Offset	Register Defination	Register Name	Number of Registers and Step Size	Initial Value
FZC_MAC+0x04008	PCS MII Status	PCS_MII_STAT	count 2 step 0x06000	
FZC_MAC+0x04010	PCS MII Advertisement	PCS_MII_ADVER	count 2 step 0x06000	
FZC_MAC+0x04018	PCS MII Partner Ability	PCS_MII_PARTNER	count 2 step 0x06000	
FZC_MAC+0x04020	PCS Configuration	PCS_CONF	count 2 step 0x06000	
FZC_MAC+0x04028	PCS State Machines	PCS_STATE	count 2 step 0x06000	
FZC_MAC+0x04030	PCS Interrupt Status	PCS_INTERRUPT	count 2 step 0x06000	
FZC_MAC+0x040A0	PCS Datapath Mode	PCS_DPATH_MODE	count 2 step 0x06000	
FZC_MAC+0x040C0	PCS Packet Counter	PCS_PKT_CNT	count 2 step 0x06000	
FZC_MAC+0x16000	MIF Bit-Bang Clock	MIF_BB_MDC		
FZC_MAC+0x16008	MIF Bit-Bang Output Data	MIF_BB_MDO		
FZC_MAC+0x16010	MIF Bit-Bang Output Enable	MIF_BB_MDO_EN		
FZC_MAC+0x16018	MIF Frame/Output	MIF_FRAME_OUTPUT_REG		
FZC_MAC+0x16020	MIF Configuration	MIF_CONFIG		
FZC_MAC+0x16028	MIF Poll Status	MIF_POLL_STATUS		
FZC_MAC+0x16030	MIF Poll Mask	MIF_POLL_MASK		
FZC_MAC+0x16038	MIF State Machine	MIF_STATE_MACHINE		
FZC_MAC+0x16040	MIF Status	MIF_STATUS		
FZC_MAC+0x16048	MIF Mask	MIF_MASK		
FZC_MAC+0x14000	Ethernet SERDES Reset	ENET_SERDES_RESET		
FZC_MAC+0x14008	Ethernet SERDES Configuration	ENET_SERDES_CFG		

TABLE 4-1 Network Interface Unit Register Map

Address Offset	Register Defination	Register Name	Number of Registers and Step Size	Initial Value
FZC_MAC+0x14010	Ethernet SERDES 0 PLL Configuration	ENET_SERDES0_PLL_CFG		
FZC_MAC+0x14018	Ethernet SERDES 0 Control	ENET_SERDES0_CONTROL		
FZC_MAC+0x14020	Ethernet SERDES 0 Test Configuration	ENET_SERDES0_TEST_CFG		
FZC_MAC+0x14028	Ethernet SERDES 1 PLL Configuration	ENET_SERDES1_PLL_CFG		
FZC_MAC+0x14030	Ethernet SERDES 1 Control	ENET_SERDES1_CONTROL		
FZC_MAC+0x14038	Ethernet SERDES 1 Test Configuration	ENET_SERDES1_TEST_CFG		
FZC_MAC+0x14040	Ethernet RGMII Configuration	ENET_RGMII_CFG		
FZC_MAC+0x14800	ESR Internal Signal	ESR_INTERNAL_SIGNALS		
FZC_MAC+0x14808	ESR Debug Selection	ESR_DEBUG_SEL		

CHAPTER 5

Neptune Software-Hardware data structures

5.1 Software data structures

Software data structures are set up during initialization of the device and maintained during operation. There are 2 major kinds of data structures:

1. Receive
2. Transmit
3. Software data formats such as descriptor words and completion words are documented in Chapters on Rx and Tx DMA

5.1.1 Transmit Descriptor Rings

One ring per DMA channel.

5.1.2 Transmit Buffer Rings

One ring per DMA Channel. Can reside anywhere in memory

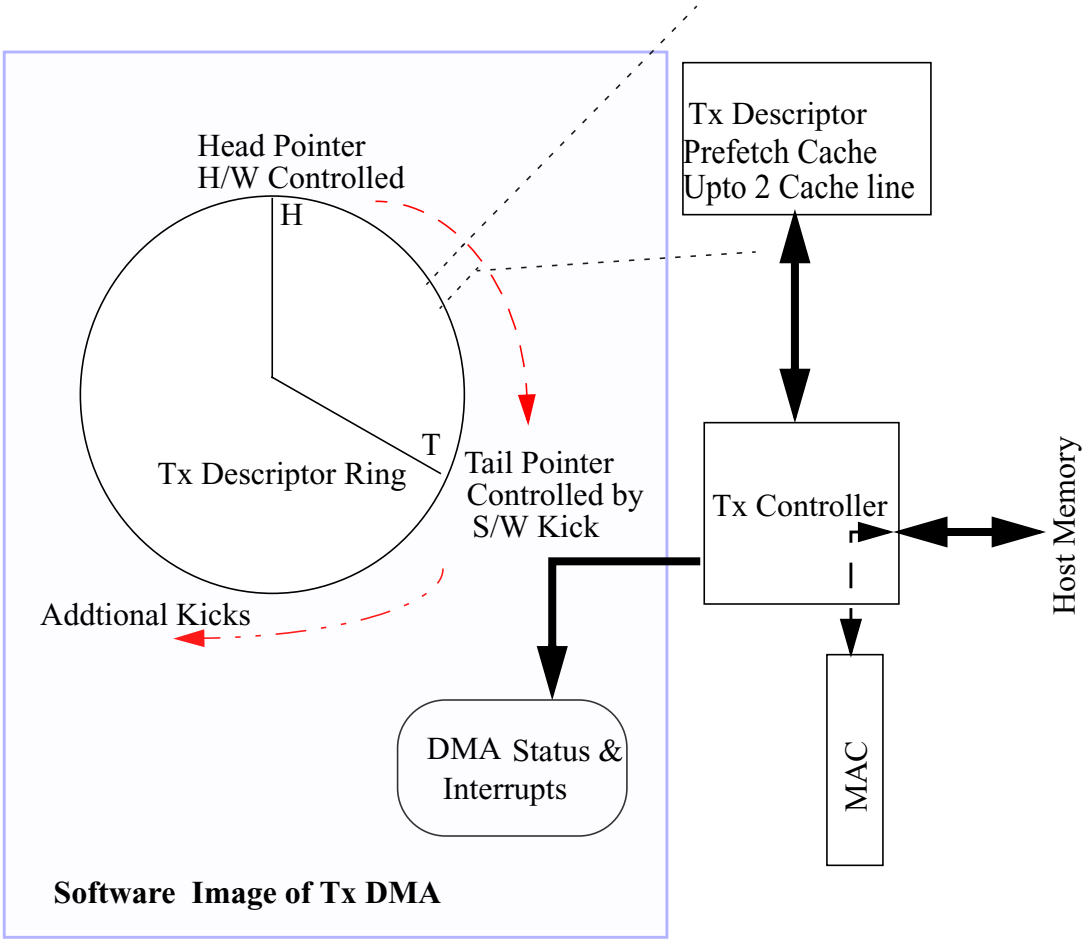

FIGURE 5-1 Transmit Descriptor Ring structure

5.1.3 Receive Descriptor Rings

One ring per DMA Channel.

5.1.4 Receive Buffer Rings

One ring per DMA Channel. Programmable sizes (Small, Medium and Large)

5.1.5 Receive Completion Rings

One ring per DMA Channel. Hardware Software handshake is involved in dynamic allocation of completion space and updates

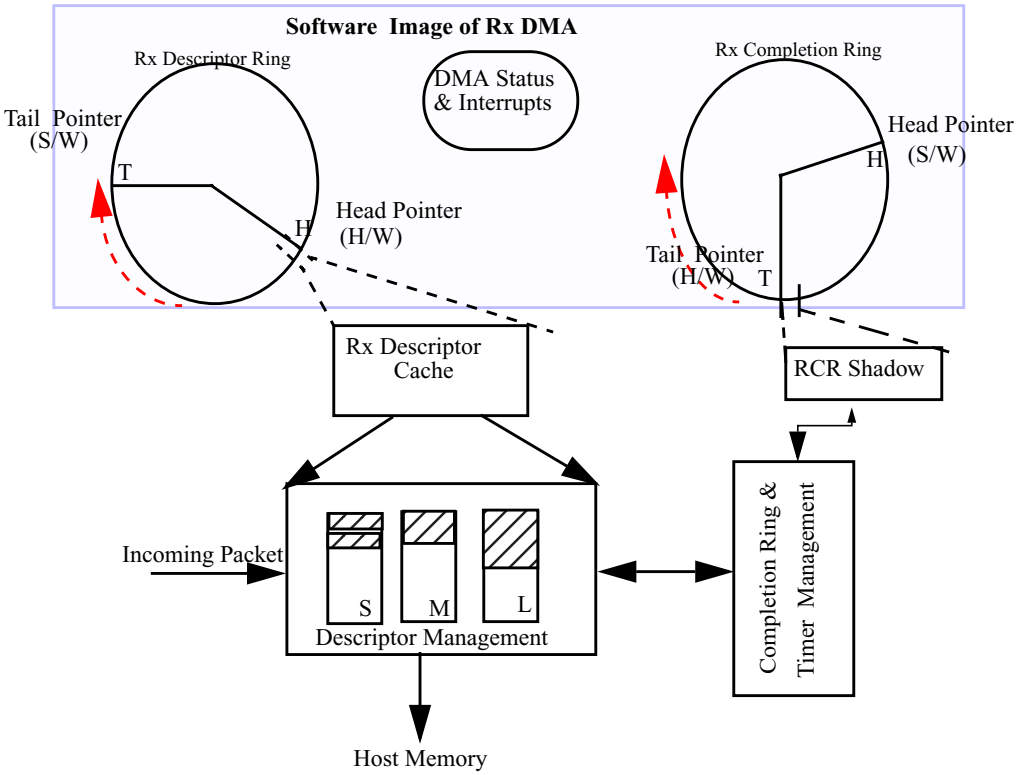

FIGURE 5-2 Receive Data Structures initialized and maintained by Software

5.2 Hardware data structures

There is no data structure per se stored on-chip, except the descriptor caches on chip to store descriptors fetched from the Host memory. The other on-chip memories include various lookup tables and FIFO storage. Their formats are documented in Chapters 10-14.

CHAPTER 11

Neptune Receive Packet Classification

11.1 Ethernet MAC Layer

The register definitions for the MAC are defined in a separate chapter. The following signals are logically available from the MAC controllers.

- Port number
- Address information: The index of the matching address, individual or group; for group addresses, the group filter result. It also indicates if it is a broadcast address.
- Layer 2 error status.

If the received frame has an address match, which can be either individual or group address, the match index and the port number are used to determine the RDC Table number. For frames that do not have an address match but pass the group filter check, an RDC table number and the preference is also specified. This information is also passed forward to the header parser to continue the classification process.

11.2 Layer 2 Classification

The Layer 2 parser will determine the following information from the frame.

1. If the frame is a VLAN packet, the VLAN ID.
2. Ethernet format, whether there is a llc/snap field.

For VLAN frames, the VLAN ID is used to index into a VLAN table, defined by the following register, to determine the RDC table number. Note that hardware does not perform parity check on PIOs. Software is required to initialize this table with proper parity value for hardware to function. For Neptune, see TABLE 11-1 for a description on seeting the values.

The following register set is for Neptune.

TABLE 11-1 Ethernet VLAN Table – ENET_VLAN_TBL (FZC_FFLP+0x00000) (count 4096 step 8)

Field	Bit Position	Initial Value	R/W	Description
—	63:18	0	RO	Reserved.
PARITY1	17	X	RW	Even parity over bits 15 to 8.
PARITY0	16	X	RW	Even parity over bits 0 to 7.
VPR3	15	X	RW	Set to 1 to indicate the preference for port 3.
VLANRDCTBLN3	14:12	X	RW	Receive DMA Channel Table number for port 3.
VPR2	11	X	RW	Set to 1 to indicate the preference for port 2.
VLANRDCTBLN2	10:8	X	RW	Receive DMA Channel Table number for port 2.
VPR1	7	X	RW	Set to 1 to indicate the preference for port 1.
VLANRDCTBLN1	6:4	X	RW	Receive DMA Channel Table number for port 1.
VPR0	3	X	RW	Set to 1 to indicate the preference for port 0.
VLANRDCTBLN0	2:0	X	RW	Receive DMA Channel Table number for port 0.

Since both the VLAN table and the MAC address table can set the preference, the following logic is used to resolve the final preference. Note that an invalid VLAN should have its vpr bit set to 0. In this case, the MAC preference is used. For example, on power-on, if no VLAN is assigned, the value of all vpr bits should be set to zero. Since we are supporting even parity, this means the entire table should be initialized to zero. If a valid VLAN is assigned, the vpr bit may be set to 1, depending on the classification requirement, and with appropriate DMA table number. The parity should be calculated accordingly.

TABLE 11-2 MAC Address / VLAN Preference Logic

mpr	vpr	Selection
0	0	MACRDCTBLN
0	1	VLANRDCTBLN
1	0	MACRDCTBLN
1	1	VLANRDCTBLN

Note that at the end of Layer 2 parsing, an RDC table number is always associated with the incoming frames the parser recognizes. Frames with errors and frames that do not have a valid MAC address will use the port default Receive DMA channel if they are not discarded. We discuss this in a later section. The Layer 2 result will be passed to the Layer 3 parser.

11.3 Layer 2/3/4 Classification

After the initial classification, hardware continues to determine the class of the received frame. Associated with each class is a Flow Key template and a TCAM Key template if the received frame is recognized by the hardware. First, based on the Ether-Type value, it will determine if the frame is one of two programmed EtherTypes (ARP or RARP), an IP (v4 or v6) frame, or others. For non-IP frames with recognized EtherType, hardware will send the class code, along with the next 11 bytes (from the EtherType) to the TCAM for a match. For IPv4 or IPv6, the parser will determine if further classification is needed, based on the Protocol ID/Next Header field and/or the `tos/dscp` byte. A number of the protocol IDs are hardwired. Software can specify up to four programmable protocol fields hardware can match, and they take precedence over hardwired match. A lower number class has a higher precedence.

TABLE 11-3 specifies the class code and describes the packet type.

TABLE 11-3 Class Code Definition

Class Code	Description	Notes
0_{16}	Hardware does not recognize the packet class.	No Key generated. This encoding indicates an invalid entry.
1_{16}	Dummy Class	Used for in-service testing. No key generated.
2_{16}	Programmable EtherType 1.	Send 11 bytes after EtherType to TCAM. No flow key generated.
3_{16}	Programmable EtherType 2.	Send 11 bytes after EtherType to TCAM. No flow key generated.

TABLE 11-3 Class Code Definition *(Continued)*

Class Code	Description	Notes
4_{16}–7_{16}	User programmable, based on protocol ID/Next Header and TOS/DSCP.	
8_{16}	TCP over IPv4.	
9_{16}	UDP over IPv4.	
A_{16}	AH or ESP over IPv4.	
B_{16}	SCTP over IPv4.	
C_{16}	TCP over IPv6.	
D_{16}	UDP over IPv6.	
E_{16}	IP v6 with Next Header AH or ESP.	
F_{16}	SCTP over IPv6.	
10_{16}	ARP	Send 11 bytes after EtherType to TCAM. No flow key generated.
11_{16}	RARP	Send 11 bytes after EtherType to TCAM. No flow key generated.
12_{16} – $1F_{16}$	Dummy Class	Used for in-service testing. No key generated.

To specify the EtherType values for class 2 and 3, the following registers are used. TABLE 11-4 describes the register for class 2; TABLE 11-5 describes the register for class 3.

TABLE 11-4 Layer 2 Class – L2_CLS (FZC_FFLP + 20000_{16}) (count 2 step 8)

Bit	Field	Initial Value	R/W	Description
63:17	—	0	RO	*Reserved*
16	vld	0	RW	Set to 1 to indicate the EtherType is valid.
15:0	etype	0	RW	EtherType value.

The following registers specify the Protocol ID (Next Header) and the tos (dscp) fields for the programmable classes 4 to 7. The first register is for class 4, second for class 5 and so on.

TABLE 11-5 Layer 3 Class – L3_CLS (FZC_FFLP + 20010_{16}) (count 4 step 8)

Bit	Field	Initial Value	R/W	Description
63:26	—	0	RO	*Reserved*
25	valid	0	RW	Set to 1 to indicate the entry is valid.
24	ipver	0	RW	0 = v4; 1 = v6.
23:16	pid	0	RW	Protocol ID or Next Header
15:8	tosmask	0	RW	Mask bits for TOS. Set a bit to 1 to specify that the corresponding bit in tos field is required in class matching.
7:0	tos	0	RW	TOS field.

Note that classes 4_{16} to F_{16} always have both a Flow key and a TCAM key associated with them. Hardware will first check if a TCAM search is required (as specified by the tsel bit below). If so, a TCAM match is initiated. If there is a hit, the associated RAM entry will specify how to continue the classification. If TCAM match is not required or there is *no* matching entry, a flow match will be initiated.

For classes 2_{16}, 3_{16}, 10_{16}, and 11_{16}, there is no flow key associated. If there is no matching entry in the TCAM, the classification process will terminate, and the packet will be directed to the default RDC associated with the RDC group.

The following registers specify how to build the TCAM Key for IP packets. If tsel bit is set to zero, hardware will not issue a TCAM match. The first register address is for class 4, and the last one for class F_{16}. For IPv6, a 4-tuple match can be initiated. For IPv4, hardware can initiate a 5-tuple match. Note that for the programmable protocol type, hardware uses the ipver bit in register L3_CLS to determine the criteria for a class. That is, class definition is IP version specific.

A v6 4-tuple plus the TOS byte consists of

- IP source or destination address
- IP protocol ID
- L4 source and destination port number or the SPI number
- TOS byte

A v4 5-tuple plus the TOS byte consists of

- IP source address
- IP destination address
- IP protocol ID
- IP source and destination port number or the SPI number
- TOS byte

TABLE 11-6 TCAM Key – TCAM_KEY (FZC_FFLP + 20030_{16}) (count 12 step 8)

Bit	Field	Initial Value	R/W	Description
63:4	—	0	RO	*Reserved*
3	disc	0	RW	Discard. Set to 1 to discard all frames of this class. The rest of the fields will not be interpreted.
2	tsel	0	RW	Set to 1 to indicate a TCAM lookup is required.
1	—	0	RO	*Reserved*
0	ipaddr	0	RW	For v6 4-tuple match, set to 1 to select source address. Set to 0 to select destination address. This bit will not be interpreted for 5-tuple match.

The objective of flow classification is to generate an offset into the RDC table to select the Receive DMA channel. The following registers specify how to build the Flow Key. The first register address is for class 4, and the last one for class F_{16}.

TABLE 11-7 Flow Key – FLOW_KEY (FZC_FFLP + 40000_{16}) (count 12 step 8)

Bit	Field	Initial Value	R/W	Description
63:10	—	0	RO	*Reserved*
9	port	0	RW	Port number, set to 1 to select.
8	l2da	0	RW	Ethernet destination MAC address. Set to 1 to select.
7	vlan	0	RW	VLAN Tag. Set to 1 to select.
6	ipsa	0	RW	IP source address. Set to 1 to select.
5	ipda	0	RW	IP destination address. Set to 1 to select.
4	proto	0	RW	Protocol IP or Next Header. Set to 1 to select.
3:2	l4_0	0	RW	Select to extract the first and second bytes after the IP header or the 5th and 6th bytes: 00 – Not select; 10 – 1st and 2nd; 11 – 5th and 6th; 01 – *Reserved*.
1:0	l4_1	0	RW	Select to extract the third and forth bytes after the IP header or the 7th and 8th bytes: 00 – Not select; 10 – 3rd and 4th; 11 – 7th and 8th; 01 – *Reserved*.

The key template is shown in TABLE 11-8. Unused fields or fields not available will be filled with zeros. For IPv4, the source/destination addresses are only 32 bits in length and will occupy the least significant 32 bits. The rest of the address field will be filled with zeros. Note that the VLAN valid field is set to 1111_2 for packets with a VLAN tag. For packets that do not have a VLAN tag, this field is set to zeros. Logically, the first bit that enters the theoretical CRC engine is bit 63 of the last 64-bit block, and the last bit is bit 0 of the first 64 bits or the first "VLAN valid" bit.

TABLE 11-8 Flow Key Template

VLAN valid	L2 DA (48bits)			VLAN ID (12)
IP Source Address {127:96}		IP Source Address {95:64}		
IP Source Address {63:32}		IP Source Address {31:0}		
IP Destination Address {127:96}		IP Destination Address {95:64}		
IP Destination Address {63:32}		IP Destination Address {31:0}		
L4-0 (16)	L4-1(16)	PID (8)	Port (2)	22 Zeros

Note that as long as the class is valid and recognized by hardware, the hash values may be reported to software through the full header if enabled.

11.3.1 TCAM Software Interface

Each TCAM entry has two fields: a key and a mask. The key field stores the pattern to be compared. The mask field, together with the key field, encodes which bit is a don't care term. Since multiple entries may generate a hit, a priority encoder on the match address(es) is needed to determine a unique hit; the lower the address, the higher the priority.

If there is a match, the match entry address is used to index into a table which stores the classification information.

Physically, the internal TCAM has only a single interface port. This port is shared by the CPU and the receive logic. The current TCAM design is 200 bits wide. For Neptune, 256 entries are supported. The description below is for programming the TCAM directly.

Each entry is composed of a key and a mask. Internally, hardware keeps a temporary register set to hold the TCAM entry. CPU then triggers a read/write/compare access to the TCAM by writing the address to the control register. Note that the same register is used to access the TCAM associated data. The compare command is for testing of the TCAM.In 32 bit mode, the Key and Mask registers are accessed least significant 32 bits first followed by the most significant 32 bits. Since side effects are triggered by writes to the the control register, there is no shadow on the Key and Mask registers.

On power-on, software needs to initialize all the TCAM entries to contain class code 0_{16}. The write and compare operations are considered *atomic*, in that the hardware should not send a request to the TCAM from the packet classifier in the middle of a write or a compare operation. This will prevent any inconsistency in the TCAM programming.

TABLE 11-9 TCAM Key 0 – TCAM_KEY_0 (FZC_FFLP + 20090_{16})

Bit	Field	Initial Value	R/W	Description
63:8	—	0	RO	*Reserved*
7:0	key	0	RW	Bits 192–199 of the TCAM entry. In Neptune, if the corresponding select bit in TCAM_KEY_MASK is cleared (0), when a read command is issued to the TCAM_CTL, a 0 will be returned, regardless of the key value used in a prior write.

TABLE 11-10 TCAM Key 1 – TCAM_KEY_1 (FZC_FFLP + 20098_{16})

Bit	Field	Initial Value	R/W	Description
63:0	key	0	RW	Bits 128–191 of the TCAM entry, or the data value in the associated RAM for TCAM results. In Neptune, if the corresponding select bit in TCAM_KEY_MASK is cleared (0), when a read command is issued to the TCAM_CTL, a 0 will be returned, regardless of the key value used in a prior write.

TABLE 11-11 TCAM Key 2 – TCAM_KEY_2 (FZC_FFLP + $200A0_{16}$)

Bit	Field	Initial Value	R/W	Description
63:0	key	0	RW	Bits 64–127 of the TCAM entry. In Neptune, if the corresponding select bit in TCAM_KEY_MASK is cleared (0), when a read command is issued to the TCAM_CTL, a 0 will be returned, regardless of the key value used in a prior write.

TABLE 11-12 TCAM Key 3 – TCAM_KEY_3 (FZC_FFLP + $200A8_{16}$)

Bit	Field	Initial Value	R/W	Description
63:0	key	0	RW	Bits 0–63 of the TCAM entry. In Neptune, if the corresponding select bit in TCAM_KEY_MASK is cleared (0), when a read command is issued to the TCAM_CTL, a 0 will be returned, regardless of the key value used in a prior write.

TABLE 11-13 TCAM Key Mask 0 – TCAM_KEY_MASK_0 (FZC_FFLP + $200B0_{16}$)

Bit	Field	Initial Value	R/W	Description
63:8	—	0	RO	*Reserved*
7:0	key_sel	0	RW	Bits 192–199 of the TCAM mask. Set 1_{16} to the corresponding bit to select the bit to be matched.

TABLE 11-14 TCAM Key Mask 1 – TCAM_KEY_MASK_1 (FZC_FFLP + $200B8_{16}$)

Bit	Field	Initial Value	R/W	Description
63:0	key_sel	0	RW	Bits 128–191 of the TCAM mask. Set 1_{16} to the corresponding bit to select the bit to be matched.

TABLE 11-15 TCAM Key Mask 2 – TCAM_KEY_MASK_2 (FZC_FFLP + $200C0_{16}$)

Bit	Field	Initial Value	R/W	Description
63:0	key_sel	0	RW	Bits 64–127 of the TCAM mask. Set 1_{16} to the corresponding bit to select the bit to be matched.

TABLE 11-16 TCAM Key Mask 3 – TCAM_KEY_MASK_3 (FZC_FFLP + $200C8_{16}$)

Bit	Field	Initial Value	R/W	Description
63:0	key_sel	0	RW	Bits 0–63 of the TCAM mask. Set 1_{16} to the corresponding bit to select the bit to be matched.

When the TCAM Control register is written to, hardware will initiate a read/write/compare to the TCAM or its associated data memory. For writes to TCAM, the data in the TCAM Key and Mask registers will be written to the TCAM at the location specified at loc. For reads from TCAM, the data at TCAM location loc will be retrieved and stored to the TCAM Key and Mask registers. In Neptune, the read command can only be used when there is no packet lookup (hardware initiated compare). This condition is satisfied when all the MACs' receive state machines are disabled. The compare command will initiate a TCAM match with the data stored in the TCAM_KEY registers. The match bit will be set to 1 if there is a TCAM match, and the matching location will be reported in the loc field. To access the TCAM associated RAM, the address of the entry is specified in loc and the TCAM_KEY_1 register is used as the data register. The status of the operation (TCAM or TCAM associated RAM) is signaled through the stat bit of the control register.

TABLE 11-17 TCAM Control – TCAM_CTL (FZC_FFLP + $200D0_{16}$)

Bit	Field	Initial Value	R/W	Description
63:21	—	0	RO	*Reserved*
20:18	rwc	0	RW	000_2 – Specifies a TCAM write; 001_2 – Specifies a TCAM read; 010_2 – For TCAM compare; 011_2 – *Reserved*; 100_2 – TCAM associated RAM write; 101_2 – TCAM associated RAM read; 110_2, 111_2 – *reserved*.
17	stat	1	RW	Status of the read or write operation. When zero is written to this bit, hardware will initiate the action, and this bit will be set to 1 when the operation completes.
16	match	0	RW	Set to 1 if there is a TCAM match for compare command. 0 otherwise.
15	—	0	RO	*Reserved*
14:10	—	0	RO	*Reserved*
9:0	loc	0	RW	TCAM or TCAM associated RAM location. For compare, this is the location of the match.

The KEY/MASK formats for TCAM are described in TABLE 11-18. Note that for IPv4 formats, a noport bit indicates whether the incoming packet is a fragment or whether the hardware cannot capture the L4 port number. The noport condition is defined as follows.

- More fragment bit is set *or* IP Offset is non-zero.

The class code *must* be programmed to be 0 to indicate an invalid entry. The reserved fields *must* be encoded as don't care terms, that is, with the corresponding key_sel bit set to 0.

TABLE 11-18 IP v4 5-Tuple Entry

Bit	Field	Comment
199:195	cls_code	Class code. Zero indicates invalid, set to the appropriate class value for the entry.
194:190	—	*Reserved*, must be set to don't care.
189:187	l2rdc_tbl_num	Layer 2 RDC Table Number
186	noport	No L4 port number.
185:112	—	*Reserved*, must be set to don't care.
111:104	tos	TOS byte.
103:96	pid	Protocol ID.
95:64	l4pt_spi	Either L4 port numbers or SPI.
63:32	ip_addr_sa	IP source address.
31:0	ip_addr_da	IP destination address.
200	*Total*	

For IPv6, the format is described in TABLE 11-19.; the EtherType format is described in TABLE 11-20.

TABLE 11-19 IP v6 4-Tuple Entry

Bit	Field	Comment
199:195	cls_code	Class code. Zero indicates invalid, set to the appropriate class value for the entry.
194:190	—	*Reserved*, must be set to don't care.
189:187	l2rdc_tbl_num	Layer 2 RDC table number.
186:176	—	*Reserved*, must be set to don't care.
175:168	tos	TOS byte.
167:160	nxt_hdr	IP v6 next header.
159:128	l4pt_spi	Source/destination port or SPI.
127:0	ip_addr	IP v6 source or destination address.
200	*Total*	

TABLE 11-20 EtherType Format

Length (bits)	Field	Comment
199:195	cls_code	Class code. Zero indicates invalid, set to the appropriate class value for the entry.
194:192	—	*Reserved*, must be set to don't care.
191:104	eframe	First 11 bits after EtherType.
103:0	—	*Reserved*, must be set to don't care.
200	*Total*	

If there is no TCAM match, the classification process will continue and the L2 RDC table number is used to select the DMA group for flow lookup.

When there is a TCAM match, the match address is used to index into the following table to retrieve the classification result. The tres field indicates whether and how to continue the classification process. Here, software may terminate the classification process and use the result specified or simply override the RDC table number. If the disc bit is set, then all frames resulting in a match in TCAM will be discarded. Hardware will also clear the age bit. (This bit may be written to 1 by software to determine if an entry has not been referenced for a period of time.)

Periodically, software should read the TCAM entries and compare with an image stored in the system memory. In addition, though the soft error probability is very low, software should check the TCAM matching criteria as indicated by the index. If the TCAM entries are used for IPv4 exact matching, software can set the v4_ecc_ck bit and include the syndrome bits in the associated data.

The ECC syndrome is calculated over the following bits.

{ associated_data{25:1}, 0_2, cam_key{103:0} }

The following is the format for the associated data.

TABLE 11-21 Format for TCAM Associated Data

Bit	Field	Description
63:42	—	*Reserved*
41:26	syndrome	ECC syndrome for IPv4 5-tuple, if V4_ECC_CK bit is not set to 1, bits 29:26 contain the even parity for the associated data. Bit 26 for bits 7:0, bit 27 for 15:8, bit 28 for 23:16, bit 29 for 25:24.
25:14	zfid	*Reserved*
13	v4_ecc_ck	Set to 1 to enable ECC error checking on the TCAM key for v4 packets. If errors are detected, the parsing operation will terminate and the RDC table from VLAN/MAC processing will be used. The offset within the table is set to zero. For v6 packets, this bit is ignored, and the syndrome bits are treated as parity. If this bit is set to zero, see description at syndrome.
12	disc	Set to 1 to discard the packet. The classification process will terminate. The value in the rest of the register will be ignored.
11:10	tres	TCAM result: 01_2 – Use the offset specified, and terminate flow lookup; 10_2 – Override the L2 RDC Table number and continue the flow lookup; 11_2 – Override the L2 RDC Table number *and* use the offset specified, terminate flow lookup; 00_2 – Use L2 RDC table number and continue the flow lookup.
9:7	rdctbl	RDC table number.
6:2	offset	Offset queue number.
1	zfvld	*Reserved*.
0	age	Hardware will clear this bit if the entry has a match during a packet lookup. PIO has no effect.

If flow lookup is terminated, zeros will be returned to software in the usr_info field in the packet header.

To filter out fragmented IPv4 packets, software needs to set one single entry in the TCAM, with only the noport bit set to 1 (*only* the respective key_sel bit should be set to 1). This entry should be put at the lowest location where IPv4 classification starts. This forces all fragmented v4 packets to match. At the corresponding TCAM associated memory location, software should specify selecting a specific default offset (with tres set to 01_2). The offset may be set to 0, which is the hardware assumed default offset. This will direct all fragmented IPv4 packets to their respective group default.

If desired, software may program a different behavior for some fragmented packets by creating entries with lower addresses than the above-mentioned location.

11.3.2 Flow Classification and Software Interface

The hashing algorithm is based on Polynomial Hashing with CRC-32C is supported. The last 4 bits of the value is used to index into the RDC table to lookup a DMA channel.

$X^{32} + X^{28} + X^{27} + X^{26} + X^{25} + X^{23} + X^{22} + X^{20} + X^{19} + X^{18} + X^{14} + X^{13} + X^{11} + X^{10} + X^9 + X^8 + X^6 + 1$

The current hardware implementation is based on an optimization that does not require "shifting" an additional *n* (the degree of the polynomial) zeros after the key as in the serial conceptual implementation. The initial value to be programmed into the hardware should be equal to the remainder of theoretical initial value (with *n* zeros appended) divided by the polynomial, using the serial implementation.

TABLE 11-22 H1 Polynomial – H1POLY (FZC_FFLP + 40060_{16})

Bit	Field	Initial Value	R/W	Description
63:32	—	0	RO	Reserved.
31:0	initval	0	RW	Initial value for Hash 1.

The following register is reserved only, and should not be accessed.

The following register set is reserved, and has no effect. The ext bit must be set to zero.

The following two registers are reserved. Software should not access these registers. Note that the access to external hash table is conditioned by the ext bit in the previous register.

TABLE 11-23 Hash Table Address – HASH_TBL_ADDR (FFLP + 00000_{16}) (count 8 step 8192)

Bit	Field	Initial Value	R/W	Description
63:24	—	0	RO	*Reserved*
23	autoinc	0	RW	*Reserved.*
22:0	addr	0	RW	*Reserved.*

TABLE 11-24 Hash Table Data – HASH_TBL_DATA (FFLP + 00008_{16}) (count 8 step 8192)

Bit	Field	Initial Value	R/W	Description
63:0	data	0	RW	*Reserved.*

11.4 Header Parser RDC Selection (FFLP)

In this section, we summarize the RDC selection by the Hardware Parser (FFLP) based on the packet header. This includes Hardware Parser error behavior and promiscuous mode. Note that this may be changed by the DMA logic if the packet has checksum or L2 CRC error. Hardware Parser does not take these errors into the processing.

In non-promiscuous mode, packets with no MAC-level match will be discarded. For packets that have a MAC-level match, an RDC group will be defined. Normally, the Hardware Parser (FFLP) will further refine the classification. If VLAN table has a hardware soft error, the MAC-level group default (the 0-th entry associated with the group select at the MAC) will be used. If TCAM logic has a hardware error, the MAC or VLAN group default will be used, depending on the preference programmed by software. In any hardware error case, there will be a signal to the DMA engine to set the completion status of the packet.

In promiscuous mode, packets with MAC level match will be processed the same way as described above. (Hardware parser may mask out the promiscuous bit by the match logic.) All other packets will be sent to the default DMA channel associated with the physical port.

The following pseudocode summarize the hardware classification process.

```
if (!MAC_addr_match) {
    MAC promiscuous mode, use per port RDC; exit;
} else {
```

```
   if (!VLAN_tagged) {
      L2_RDC_table_num = MAC_RDC_table_num; goto L2-3-4;
   } else {
      use (port, VLANID) as index to VLAN table to lookup
      VLAN_RDC_table_num;
      if (VLAN_parity_err) {
         L2_RDC_table_num = MAC_RDC_table_num; goto L2-3-4; }
      if (VPR==1) {
         L2_RDC_table_num = VLAN_RDC_table_num;
      } else {
         L2_RDC_table_num = MAC_RDC_table_num; }
      }
   }

label - L2-3-4:

   if (!Class_match) {
      RDC_table_num = L2_RDC_table_num; offset = 0; exit; }
   if (4<=class<=15) {
      if (DISC) { set drop_pkt = 1; exit; }
      if (TCAM_lookup_not_required) goto flow_1;
   }
   start_TCAM_search;
   if (!TCAM_match) goto flow_1;
   read_TCAM_assoc_data;
   if (TCAM_assoc_dat_DISC==1) { set drop_pkt = 1; exit; }
   if (TCAM_assoc_dat_TRES[1]==1) {
      RDC_table_num = L2_RDC_table_num;
   } else { RDC_table_num = TCAM_assoc_dat_table_num; }
   if (TCAM_assoc_dat_TRES[0]==1) {
      offset = TCAM_assoc_dat_RDC_table_offset; exit; }
   if (!(4<=class<=15)) { offset = 0; exit;}
   goto flow_2;

label - flow_1:

   RDC_table_num = L2_RDC_table_num;
   if (!(4<=class<=15)) { offset = 0; exit;}

label - flow_2:

offset = H1[3:0];exit;

}
```

11.5 Checksum Offload

The hardware supports TCP/UDP payload checksum in the receive datapath for nonfragmented packets.

11.6 Receive Packet Header Format and Alignment

Receive Header Format Classification result will be prepended to the packet and stored in the packet buffer. The formats are shown below. Note that the entire full header is 18 bytes. It is divided into Header 0 and Header 1. Software may select only the two-byte Header 0 to be sent as header, by clearing the full_hdr bit in registers for configuring the receive DMAs. The full_hdr bit setting is per DMA. The following formats are defined in address order. B0 is in low address.

TABLE 11-25 Receive Packet Header 0 Format

Field	Position		Description
	Byte	Bit	
inputport	0	7:6	*Physical* input port number.
maccheck	0	5	Set to 1 to indicate MAC address check passed. For individual address, this indicates that the destination MAC address matches one of the programmable value. For group address, the corresponding multicast filter entry is set.
class	0	4:0	Class value.
vlan	1	7	Set to 1 if vlan field is detected. Set to 0 if l2par = 0.
llcsnap	1	6	Set to 1 if the L2 frame contains an LLC/SNAP header. Set to 0 if l2par = 0.
noport	1	5	Set to 1 to indicate hardware cannot reach the L4 ports or SPI for IPSec.
badip	1	4	This bit will be forced to zero if class does not indicate an IP packet (see class code description). If this bit is set, it indicates hardware cannot process the IP packet. This includes the following case(s): IHL < 5. In this case, the packet will be sent to the Layer 2 default DMA.
tcamhit	1	3	Set to 1 if a TCAM search is successful.
tres	1	2:1	Same as tres bits in TCAM_RES register. If tcamhit bit is zero, tres bits will be set to zero.
tzfvld	1	0	Same bit as zfvld bit in tcam_res register; always set to zero if tcamhit is 0.

TABLE 11-26 Receive Packet Header 1 Format

Field	Position		Description
	Byte	Bit	
hwrsvd	0		*Reserved* for HW use.
tcammatch	1		TCAM match index. If class ≠ 0 and tcamhit = 1, this is the TCAM match index; set to zero otherwise.
—	2	7:6	*Reserved* for hardware.
hashit	2	5	Set to 1 for Hash hit.
exact	2	4	Set to 1 to indicate an exact match in hash table lookup is resulted. Set to 0 if hashit = 0.
hzfvld	2	3	Set to 1 if Hash table entry indicates a zero copy flow ID is available.
hashsidx	2	2:0	Subindex into the Hash Table Entry, set to zero if hashit = zero.
—	3	3	Reserved for zero copy function.
—	4	7:4	*Reserved* for hardware use.
zflowid	4	3:0	Bits 11:8 of zero copy flow ID. Valid if either tzfvld or hzfvld is set. Set to 0 otherwise. For TCAM match, it is the TCAM entry number; for hash table, it is the value stored in the hash table. The unused higher order bits are set to zero.
zflowid	5		Bits 7:0 of zero copy flow ID.
hashval2	6		Bits 15:8 of hash value, H2. Valid when class indicates an IP packet, zero otherwise.
hashval2	7		Bits 7:0 of hash value, H2. Valid when class indicates an IP packet, zero otherwise.
—	8	7:4	*Reserved* for hardware.
hashval1	8, 9, 10	3:0	Hash value, H1. Valid when class indicates an IP packet, zero otherwise. Bits 19:16 are in byte 10; 15:8 in byte 11; 7:0 in byte 12.
usrdata	11:15		Bits 39:0 of user data, valid if tres is either 00_2 or 10_2. Set to zero otherwise. 39:32 in byte 13; 31:24 in byte14; 23:16 in byte 15; 15:8 in byte 16; 7:0 in byte 17.

The following shows an example of a non-jumbo packet format in address order. A packet buffer is at least 64 bytes naturally aligned. There may be multiple of 64-byte blocks reserved by software. The packet header starts after the reserved area and is immediately followed by the packet. As mentioned earlier, the packet header can be 2 bytes or 18 bytes. In either case, the IP packet will always be 4-byte aligned. The multiple 64-byte offset will be discussed in a later section.

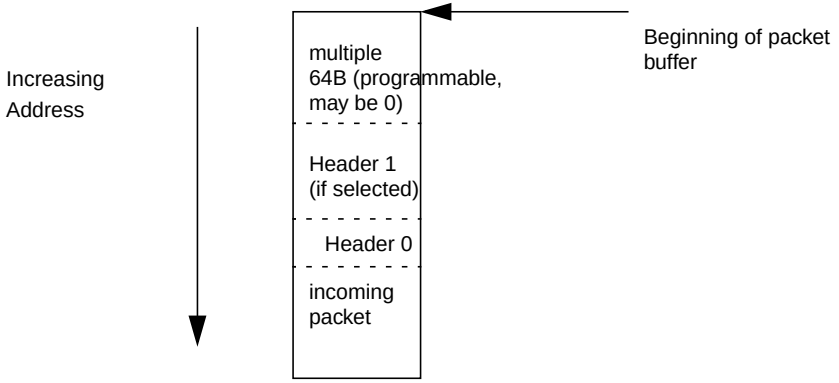

FIGURE 11-1 Packet Buffer Layout

11.7 FFLP Hardware Control Registers

The following registers are used to configure the FFLP.

TABLE 11-27 FFLP Configuration 1 – FFLP_CFG_1 (FZC_FFLP + 20100_{16})

Bit	Field	Initial Value	R/W	Description
63:27	—	0	RO	*Reserved*
26	tcam_dis	0	RW	Set to 1 to disable TCAM, all TCAM search results will be treated as no match.
25:23	pio_dbg_sel	0	RW	000_2 –101_2: debug_data 110_2: debug_training_vector 111_2: ~fflp_debug_port
22	pio_fio_rst	0	RW	*Reserved*.
21:20	pio_fio_lat	0	RW	*Reserved*.
19:16	camlat	0	RW	CAM search latency, Should be set to 4.
15:12	camratio	0	RW	CAM access ratio, similar to bits 11:8.
11:8	fcramratio	0	RW	Reserved.
7:4	fcramoutdr	0	RW	*Reserved.*
3	fcramqs	0	RW	*Reserved.*
2	errordis	0	RW	Set to 1 to disable error check in TCAM.
1	fflpinitdone	0	RW	FFLP initialization done.
0	llcsnap	0	RW	Set to 1 to enable LLCSNAP packets.

TABLE 11-28 FFLP Debug Training Vector – FFLP_DBG_TRAIN_VCT (FZC_FFLP + 20148_{16})

Bit	Field	Initial Value	R/W	Description
63:32	—	0	RO	*Reserved*
31:0	vector	0	RW	Debug training vector.

TABLE 11-29 TCP Control Flag Mask – TCP_CFLAG_MSK (FZC_FFLP + 20108_{16})

Bit	Field	Initial Value	R/W	Description
63:12	—	0	RO	*Reserved*
11:0	mask	0	RW	Set to 1 to mask out the correspondent control bit. This field includes 6 control bits and 6 reserved bits in TCP header. This bit selects the bit to be propagated to DMA controller. This should be set to select the sync bit only for the DMA to pick a different WRED parameter.

TABLE 11-30 FCRAM Refresh Timer – FCRAM_REF_TMR (FZC_FFLP + 20110_{16})

Bit	Field	Initial Value	R/W	Description
63:32	—	0	RO	*Reserved*
31:16	max	0	RW	*Reserved.*
15:0	min	0	RW	*Reserved.*

The following registers are reserved. Access has no effect.

TABLE 11-31 FCRAM Controller Address – FCRAM_FIO_ADDR (FZC_FFLP + 20118_{16})

Bit	Field	Initial Value	R/W	Description
63:8	—	0	RO	*Reserved*
7:0	addr	0	RW	*Reserved.*

TABLE 11-32 FCRAM Controller Data – FCRAM_FIO_DAT (FZC_FFLP + 20120_{16})

Bit	Field	Initial Value	R/W	Description
63:16	—	0	RO	*Reserved*
15:0	data	0	RW	*Reserved.*

TABLE 11-33 FCRAM PHY Read Latency – FCRAM_PHY_RD_LAT (FZC_FFLP + 20150_{16})

Bit	Field	Initial Value	R/W	Description
63:8	—	0	RO	*Reserved*
7:0	lat	0	RO	*Reserved.*

11.8 Error Registers

The FFLP_VLAN_PAR_ERR register logs the VLAN table parity error. Software writes 0's to clear. This register logs errors only during packet lookup.

TABLE 11-34 VLAN Parity Error – FFLP_VLAN_PAR_ERR (FZC_FFLP + 08000_{16})

Bit	Field	Initial Value	R/W	Description
63:32	—	0	RO	*Reserved*
31	err	0	RW	Set to 1 to indicate an error; software writes 0 to clear.
30	m_err	0	RW	Set to 1 to indicate multiple error; software writes 0 to clear.
29:18	addr	0	RW	Address within the block where error occurred.
17:0	data	0	RW	Data read from memory.

The following register logs the TCAM error during packet lookup. The error log needs to be cleared by software.

TABLE 11-35 TCAM Error – TCAM_ERR (FZC_FFLP + $200D8_{16}$)

Bit	Field	Initial Value	R/W	Description
63:32	—	0	RO	*Reserved*
31	err	0	RW	Set to 1 to indicate TCAM error, either in the associated data or TCAM when detected. Software writes 0 to clear.
30	p_ecc	0	RW	Set to 1 to indicate if this is a v4 ECC error; 0 for parity error
29	mult	0	RW	Set to 1 to indicate multiple lookup error
28:24	—	0	RO	*Reserved*
23:16	addr	0	RW	Entry where the error occurs.
15:0	syndrome	0	RW	Hardware-calculated syndrome or parity value for the first error.

The FFLP Error Mask register defines which event will be reported. A logical **or** of the errors with the mask bit cleared will be signaled to the PIO block for error reporting.

TABLE 11-36 FFLP Error Mask – FFLP_ERR_MSK (FZC_FFLP + 20140_{16})

Bit	Field	Initial Value	R/W	Description
63:11	—	0	RO	*Reserved*
10:3	hsh_tbl_dat	FF_{16}	RW	
2	hsh_tbl_lkup	1	RW	
1	tcam	1	RW	TCAM lookup. Set to 0 to enable the event reporting.
0	vlan	1	RW	*Reserved*

The following registers are reserved and should always set at the default value.

TABLE 11-37 Hash Table Data Error log – HASH_TBL_DATA_LOG (FFLP + 00010_{16}) (count 8 step 8192)

Bit	Field	Initial Value	R/W	Description
63:32	—	0	RO	*Reserved*
31	err	0	RW	*Reserved*
30:8	addr	0	RW	*Reserved*
7:0	syndrome	0	RW	*Reserved*

TABLE 11-38 Hash Table Lookup Error Log 1 – HASH_LOOKUP_ERR_LOG1 (FZC_FFLP + $200E0_{16}$)

Bit	Field	Initial Value	R/W	Description
63:4	—	0	RO	*Reserved*
3	err	0	RW	*Reserved*
2	mult_lk	0	RW	*Reserved*
1	cu	0	RW	*Reserved*
0	mult_bit	0	RW	*Reserved*

TABLE 11-39 Hash Table Lookup Error Log 2 – HASH_LOOKUP_ERR_LOG2 (FZC_FFLP + $200E8_{16}$)

Bit	Field	Initial Value	R/W	Description
63:31	—	0	RO	*Reserved*
30:11	h1	0	RW	*Reserved*
10:8	subarea	0	RW	*Reserved*
7:0	syndrome	0	RW	*Reserved*

TABLE 11-40 FCRAM Error Test 0 – FCRAM_ERR_TST0 (FZC_FFLP + 20128_{16})

Bit	Field	Initial Value	R/W	Description
63:8	—	0	RO	*Reserved*
7:0	synd	0	RW	*Reserved*

TABLE 11-41 FCRAM Error Test 1 – FCRAM_ERR_TST1 (FZC_FFLP + 20130_{16})

Bit	Field	Initial Value	R/W	Description
63:32	—	0	RO	*Reserved*
31:0	dat	0	RW	*Reserved*

TABLE 11-42 FCRAM Error Test 2 – FCRAM_ERR_TST2 (FZC_FFLP + 20138_{16})

Bit	Field	Initial Value	R/W	Description
63:32	—	0	RO	*Reserved*
31:0	dat	0	RW	*Reserved*

APPENDIX **A**

Neptune Initialization sequence

A.1 Introduction

This PRM chapter describes the Neptune Transmit and receive initialization and reconfiguration sequences.

Each sequence reflects the steps and order in which the individual Neptune blocks have to be initialized and enabled to achieve the successful transmit and/or receive of Ethernet packets.

Each step provides a link to a paragraph with a detailed description of the registers that have to be programmed in order to complete the step. <click on paragraph>

Registers that are not required to be configured for intialization purposes like error interrupts etc, are not covered in this document.

Register initialization is done by PIO read and write request issued across the Neptune PCIe interface. As a result, a PCIe link connection has to be established before the Neptune initialization sequence can commence. The description of the PXIe link configuration sequence is outside the scope of this document.

The following Intialization and reconfiguration sequences are included:

- Transmit Power-on-reset initialization sequence A.2
- Receive Power-on-reset initialization sequence A.3
- Loopback POR Initialization sequence A.4
- RX and TX Port Reconfiguration A.5
- Rx DMA channel reconfiguration A.6
- Tx DMA channel reconfiguration A.7
- TXC reconfiguration A.8

The detailed description of the blocks can be found in:

Block level initialization A.9

A.2 Transmit Power-on-reset initialization sequence

- Identify Board/Card
- Initialize MIF config register A.9.1.3
- Ethernet Serdes Initialization A.9.1
  - Initialize SERDES through MIF A.9.1.4
  - Program PLL parameters A.9.1.5
  - Initialize Transceiver A.9.1.6
  - Wait for PLL lock A.9.1.7
- MAC Initialization A.9.2
  - Initialize XPCS/PCS A.9.2.2
  - Select MAC clock source and mode A.9.2.1
  - Soft Reset TxMAC A.9.2.3
  - Initialize TxMAC A.9.2.7
- TxDMA Initialization A.9.6
  - Configure TXC A.9.6.5
  - Enable TXC A.9.6.6
  - Bind TxDMA channel to Port A.9.6.7
  - Initialize TxDMA A.9.6.8
- Enable TxMAC A.9.2.9
- Enable TxDMA A.9.6.9

A.3 Receive Power-on-reset initialization sequence

- Identify Board/Card
- Initialize MIF config register A.9.1.3
- RxDMA Initialization A.9.5
  - Initialize RxDMA A.9.5.4
- FFLP Initialization (classification) A.9.4
  - Initialize FFLP A.9.4.2
  - Configure FFLP A.9.4.3 (recommended)
- ZCP Initialization A.9.7
- Ethernet Serdes Initialization A.9.1
  - Initialize SERDES through MIF A.9.1.4
  - Program PLL parameters A.9.1.5
  - Initialize Transceiver A.9.1.6
  - Wait for PLL lock A.9.1.7
- MAC Initialization A.9.2
  - Select MAC clock source and mode A.9.2.1
  - Initialize XPCS/PCS A.9.2.2
  - Soft Reset RxMAC A.9.2.4
  - Initialize RxMAC A.9.2.8
- Enable RxDMA A.9.5.5
- Enable IPP Port A.9.3.5
- Enable RxMAC A.9.2.10

A.4 Loopback POR Initialization sequence

- Identify required Loopback mode
- Complete Receive Power-on-reset initialization sequence A.3
- Complete Transmit Power-on-reset initialization sequence A.2

A.5 RX and TX Port Reconfiguration

A.5.1 Reconfigure Tx Port

- Disable TxDMA A.9.6.10
- Disable TXC ports A.9.6.11
- Stop TxMAC A.9.2.5
- Soft Reset TxMAC A.9.2.3
- Initialize TxMAC A.9.2.7
- Configure TXC A.9.6.5
- Enable TXC A.9.6.6
- Initialize TxDMA A.9.6.8
- Enable TxMAC A.9.2.9
- Enable TxDMA A.9.6.9

A.5.2 Reconfigure Rx Port

- Stop RxMAC A.9.2.6
- Drain IPP Port A.9.3.6
- For each associated channel: Stop and reset RxDMA A.9.5.3
- Reset ZCP Control Fifo A.9.7.1
- Soft reset IPP A.9.3.3
- Soft Reset RxMAC A.9.2.4
- Re-Configure FFLP A.9.4.3 (if required)
- Configure ZCP-RDC table A.9.7.2 (if required)
- Initialize RxMAC A.9.2.8
- Enable IPP Port A.9.3.5
- Enable RxMAC A.9.2.10

A.6 Rx DMA channel reconfiguration

A.6.1 Bind Rx DMA channel

Channel in POR state:

- Initialize RxDMA A.9.5.4
- Enable RxDMA A.9.5.5

Channel not in POR state

- Reset RxDMA A.9.5.1
- Initialize RxDMA A.9.5.4
- Enable RxDMA A.9.5.5

A.6.2 Remove Rx DMA channel

- Disable RxDMA A.9.5.6

A.6.3 Reconfigure Rx DMA channel

- Reset RxDMA A.9.5.1
- Initialize RxDMA A.9.5.4
- Reconfigure RxDMA
- Enable RxDMA A.9.5.5

A.7 Tx DMA channel reconfiguration

A.7.1 Bind Tx DMA channel

- Soft Reset TxDMA A.9.6.2
- Bind TxDMA channel to Port A.9.6.7
- Initialize TxDMA A.9.6.8
- Enable TxDMA A.9.6.9

A.7.2 Remove Tx DMA channel

- Disable TxDMA A.9.6.10

A.7.3 Reconfigure Tx DMA channel

- Disable TxDMA A.9.6.10
- Re Bind TxDMA channel to Port A.9.6.7
- Soft Reset TxDMA A.9.6.2
- Re Initialize TxDMA A.9.6.8
- Re configure TxDMA
- Enable TxDMA A.9.6.9

A.8 TXC reconfiguration

A.8.1 Add TXC port

- Configure TXC A.9.6.5

A.8.2 Remove TXC port

- Disable TxDMA A.9.6.10
- Disable TXC ports A.9.6.11

A.9 Block level initialization

A.9.1 Ethernet Serdes Initialization

This chapter describes the Neptune Serdes initialization sequence.

A.9.1.1 Hw Serdes Reset

- Global hardware reset, the niu_reset_l
  - Applies to all the SERDES
  - Initiates Serdes Reset sequence. Leaves all Serdes registers in the initial POR state.
  - All channels are enabled
  - PLL is locked
  - Tx sends IDLE packets.

A.9.1.2 Soft Serdes reset

Serdes support 2 kinds of soft reset:

PIO:

- Assert serdes_reset_1 ENET_SERDES_RESET[1] = 1_2 (FZC_MAX+14000_{16})

- Assert serdes_reset_0 ENET_SERDES_RESET[0] = 1_2 (FZC_MAX+14000_{16})

The Serdes remain in reset state as long as the serdes reset bits are asserted. To exit the reset state, the serdes_reset_1 and serdes_reset_0 fiedls have to be de-asserted by software.

MDIO:

- Assert tx/rx reset_<0-3>_<a-d> fields in RX_TX_RESET_CONTROL for each lane.

The fields self-clear when soft reset sequence is completed.

- Poll serdes_rdy ESR_INTERNAL_SIGNALS[] = 1_2 (FZC_MAC+14800_{16})

A.9.1.3 Initialize MIF config register

Default	MIF_CONFIG[63:0]	= 0080_{16}	(FZC_MAC+16020_{16})
ATCA	MIF_CONFIG[63:0]	=	(FZC_MAC+16020_{16})
Indirect	MIF_CONFIG[63:0]	=	(FZC_MAC+16020_{16})

A.9.1.4 Initialize SERDES through MIF

- Applies to Serdes

Set the following registers through MIF :

A.9.1.5 Program PLL parameters

(not required after POR incase of 10G speed):

- Put Serdes in reset by setting
- Set PLL speed:
- Remove reset by setting

A.9.1.6 Initialize Transceiver

A.9.1.7 Wait for PLL lock

Poll ESR_INTERNAL_SIGNALS

A.9.2 MAC Initialization

A.9.2.1 Select MAC clock source and mode

- **XMAC**

Default XMAC_CONFIG[63:0] = $02C20004_{16}$ ($FZC\_MAC+00060_{16}$)

MIF_CONFIG[16] = 0_2 ($FZC\_MAC+16020_{16}$)

10G XAUI	pcs_bypass	MAC_XIF_CONFIG[30]	= 0_2
	xpcs_bypass	MAC_XIF_CONFIG[29]	= 0_2
	mii_gmii_mode	MAC_XIF_CONFIG[28:27]	= 00_2
	loopback	MAC_XIF_CONFIG[25]	= 0_2
	sel_por_clk_src	MAC_XIF_CONFIG[23]	= 0_2
	atca_GE	MIF_CONFIG[16]	= 0_2
1G Optical	pcs_bypass	MAC_XIF_CONFIG[30]	= 0_2
	xpcs_bypass	MAC_XIF_CONFIG[29]	= 0_2
	mii_gmii_mode	MAC_XIF_CONFIG[28:27]	= 01_2
	loopback	MAC_XIF_CONFIG[25]	= 0_2
	sel_por_clk_src	MAC_XIF_CONFIG[23]	= 0_2
	atca_GE	MIF_CONFIG[16]	= 0_2
1G RGMII	pcs_bypass	MAC_XIF_CONFIG[30]	= 1_2
	xpcs_bypass	MAC_XIF_CONFIG[29]	= 0_2
	mii_gmii_mode	MAC_XIF_CONFIG[28:27]	= 01_2
	loopback	MAC_XIF_CONFIG[25]	= 0_2
	sel_por_clk_src	MAC_XIF_CONFIG[23]	= 0_2
	atca_GE	MIF_CONFIG[16]	= 0_2

1G ATCA	pcs_bypass (1G)	MAC_XIF_CONFIG[30]	= 0_2
	xpcs_bypass (10G)	MAC_XIF_CONFIG[29]	= 0_2
	mii_gmii_mode	MAC_XIF_CONFIG[28:27]	= 01_2
	loopback	MAC_XIF_CONFIG[25]	= 0_2
	sel_por_clk_src	MAC_XIF_CONFIG[23]	= 0_2
	atca_GE	MIF_CONFIG[16]	= 1_2

- **BMAC**

Default MAC_XIF_CONFIG[63:0] = $4A_{16}$ (FZC_MAC+$0C078_{16}$)

PCS_DPATH_MODE[63:0]= 0_{16} (FZC_MAC+040A0 $_{16}$)

1G Optical	sel_clk_25mhz	MAC_XIF_CONFIG[7]	= 0_2
	gmii_mode	MAC_XIF_CONFIG[3]	= 1_2
	phy_mode	PCS_DPATH_MODE[1]	= 0_2

1G RGMII	sel_clk_25mhz	MAC_XIF_CONFIG[7]	= 0_2
	gmii_mode	MAC_XIF_CONFIG[3]	= 1_2
	phy_mode	PCS_DPATH_MODE[1]	= 1_2

Note – Any changes to the CONFIG registers should be followed a MAC software reset.

- **Loopback XMAC**

10G XAUI	loopback	XMAC_CONFIG[25]	$= 1_2$
	xpcs_loopback	BASE10G_CONTROL1[14]	$= 0_2$
	atca_GE	MIF_CONFIG[16]	$= 0_2$
10G XPCS	loopback	XMAC_CONFIG[25]	$= 0_2$
	xpcs_loopback	BASE10G_CONTROL1[14]	$= 1_2$
	atca_GE	MIF_CONFIG[16]	$= 0_2$
1G Optical	loopback	XMAC_CONFIG[25]	$= 1_2$
	xpcs_loopback	BASE10G_CONTROL1[14]	$= 0_2$
	atca_GE	MIF_CONFIG[16]	$= 0_2$
1G RGMII	loopback	XMAC_CONFIG[25]	$= 1_2$
	xpcs_loopback	BASE10G_CONTROL1[14]	$= 0_2$
	atca_GE	MIF_CONFIG[16]	$= 0_2$

- **Loopback BMAC**:

1G Opt	loopback	MAC_XIF_CONFIG[1]	$= 1_2$
1G RGMII	loopback	MAC_XIF_CONFIG[1]	$= 1_2$

A.9.2.2 Initialize XPCS/PCS

- 10G Fiber reset XPCS BASE10G_CONTROL_1 = 5020_{16} (FZC_MAC+02000_{16})

- 1G Fiber PCSInternal PCS_DPATH_MODE[63:0] = 0000_{16} (FZC_MAC+$040A0_{16}$)
  reset PCS PCS_MII_CTL[63:0] = 9040_{16} (FZC_MAC+04000_{16})
  enable PCS PCS_CONFIG[63:0] = 0001_{16} (FZC_MAC+04020_{16})

A.9.2.3 Soft Reset TxMAC

- **XMAC**

Set	TxMacSoftRst	XTxMAC_SW_RST[0]	= 1_2	(FZC_MAC+00000_{16})
	TxMacRegRst	XTxMAC_SW_RST[1]	= 1_2	
Poll		XTxMAC_SW_RST[1:0]	= 00_2	(done indicator)

- **bMAC**

Set	txmac_sw_rst	BTxMAC_SW_RST[0]	= 1_2	(FZC_MAC+$0C000_{16}$)
Poll	txmac_sw_rst	BTxMAC_SW_RST[0]	= 0_2	(done indicator)

A.9.2.4 Soft Reset RxMAC

- **XMAC**

Set	RxMacSoftRst	XRxMAC_SW_RST[0]	= 1_2	(FZC_MAC+00008_{16})
	RxMacRegRst	XRxMAC_SW_RST[1]	= 1_2	

Poll XRxMAC_SW_RST[1:0] = 0_2 (done indicator)

- **bMAC**

Set rxmac_sw_rst BRxMAC_SW_RST[0] = 1_2 (FZC_MAC+$0C008_{16}$)

Poll rxmac_sw_rst BRxMAC_SW_RST[0] = 0_2 (done indicator)

A.9.2.5 Stop TxMAC

- **XMAC**

Set tx_enable XMAC_CONFIG[0] = 0_2 (FZC_MAC+00060_{16})

Poll xtlm_state XMAC_SM_REG[2:0] = 000_2 (FZC_MAC+$001A8_{16}$)

- **bMAC**

Set tx_enable TxMAC_CONFIG[0] = 0_2 (FZC_MAC+$0C060_{16}$)

Poll xtlm_state BMAC_SM_REG[19:16] = 0000_2 (FZC_MAC+$0C3A0_{16}$)

A.9.2.6 Stop RxMAC

- **XMAC**

Set rx_enable XMAC_CONFIG[8] = 0_2 (FZC_MAC+00060_{16})

Poll xrlm_state XMAC_SM_REG[8] = 0_2 (FZC_MAC+$001A8_{16}$)

■ **bMAC**

Set rx_enable	RxMAC_CONFIG[0]	= 0_2	(FZC_MAC+0C068$_{16}$)
Poll rtlm_state (FZC_MAC+0C3A0$_{16}$)	BMAC_SM_REG[26:23]	= 0000_2	

Note – The Rx Stop sequence grafully terminates receiving data at a packet boundary, and completes the transfer of outstanding packets to the IPP.

A.9.2.7 Initialize TxMAC

■ **XMAC**

Set StretchConstant	XMAC_IPG[32:21]	Def = 1_{16}	(FZC_MAC+080$_{16}$)
StretchRatio	XMAC_IPG[20:16]	Def = 0D$_{16}$	
Ipg_value1	XMAC_IPG[15:8]	Def = 0A$_{16}$	
ipg_value	XMAC_IPG[2:0]	Def = 3_{16}	
Set SlotTime	XMAC_MIN[17:10]	Def = 08_{16}	(FZC_MAC+088$_{16}$)
TxMinPacketSize	XMAC_MIN[9:3]	Def = 08_{16}	
SetMaxFrameSize	XMAC_MAX[13:0]	Def = 1518_{10}	(FZC_MAC+090$_{16}$)

■ **BMAC**

Set min_pkt_size (FZC_MAC+C0A0$_{16}$)	BMAC_MIN[9:0]	Rec = 040_{16}	
Set MaxFrameSize (FZC_MAC+C0A8$_{16}$)	BMAC_MAX[14:0]	Rec = 1518_{10}	
MaxBurstSize	BMAC_MAX[30:16]	Rec = 1518_{10}	
Set control_type (FZC_MAC+C0C8$_{16}$)	MAC_CTRK_TYPE	Rec = 8808_{16}	
Set pa-size	MAC_PA_SIZE[9:0]	Rec = 007_{16}	(FZC_MAC+C0B0$_{16}$)

A.9.2.8 Initialize RxMAC

- **XMAC**

Set MacUniqueAddr<0-2>XMAC_ADDR<0-2>[15:0] (FZC_MAC+$0A0_{16}$)

Set StretchConstant	XMAC_IPG[32:21]	Def = 1_{16}	(FZC_MAC+080_{16})
StretchRatio	XMAC_IPG[20:16]	Def = $0D_{16}$	
Ipg_value1	XMAC_IPG[15:8]	Def = $0A_{16}$	
ipg_value	XMAC_IPG[2:0]	Def = 3_{16}	
Set SlotTime	XMAC_MIN[17:10]	Def = 08_{16}	(FZC_MAC+088_{16})
RxMinPacketSize	XMAC_MIN[29:20]	Def = 040_{16}	
SetMaxFrameSize	XMAC_MAX[13:0]	Def = 1518_{10}	(FZC_MAC+090_{16})

Set alt_addr_cmp_en[15:0] FZC_MAC+208_{16})	XMAC_ADDR_CMPEN	
Set mac_alt_addr<0-15>_<0-2> FZC_MAC+218_{16})	XMAC_ADDR<3-50>	
Set mac_addr_filter_<0-2> FZC_MAC+818_{16})	XMAC_ADDR_FILT<0-2>	
Set mac_addr_filter_mask12/00 $_{16}$)	XMAC_ADDR_FILT12/00_MASK	FZC_MAC+830
Set xmac_hash_<0-15> FZC_MAC+840_{16})	XMAC_HASH_TBL<0-15>	
Set MPR FZC_MAC+900_{16})	XMAC_HOST_INFO<0-19>[8]	
MACRDCTBLN	XMAC_HOST_INFO<0-19>[2:0]	

- **BMAC**

Set BMacUniqueAddr<0-2	BMAC_ADDR<0-2>[15:0]	FZC_MAC+$C108_{16}$)

Set min_pkt_size	BMAC_MIN[9:0]	Rec = 040_{16}	
FZC_MAC+C0A0$_{16}$)			
Set MaxFrameSize	BMAC_MAX[14:0]	Rec = 1518_{10}	
FZC_MAC+C0A8$_{16}$)			
MaxBurstSize	BMAC_MAX[30:16]	Rec = 1518_{10}	
Set control_type	MAC_CTRK_TYPE	Rec = 8808_{16}	FZC_MAC+C0C8$_{16}$)

Set alt_addr_cmp_en[15:0]	BMAC_ALTAD_CMPEN	FZC_MAC+C3F8$_{16}$)
Set mac_alt_addr<0-6>_<0-2>	MAC_ADDR<3-23>	FZC_MAC+C118$_{16}$)
Set mac_addr_filter_<0-2>	MAC_ADD_FILT<0-2>	FZC_MAC+C298$_{16}$)
Set mac_addr_filter_mask12/00	MAC_ADDR_FILT12/00_MASK	
FZC_MAC+C2B0$_{16}$)		
Set xmac_hash_<0-15>	MAC_HASH_TBL<0-15>	FZC_MAC+C2C0$_{16}$)
Set MPR	bMAC_HOST_INFO<0-19>[8]	FZC_MAC+C400$_{16}$)
MACRDCTBLN	bMAC_HOST_INFO<0-19>[2:0]	

A.9.2.9 Enable TxMAC

- **XMAC**

set tx_output_en	XMAC_CONFIG[24]	= 1_2	(FZC_MAC+00060$_{16}$)
tx_enable	XMAC_CONFIG[0]	= 1_2	

- **BMAC**

set tx_output_en	MAC_XIF_CONFIG[0]	= 1_2
(FZC_MAC+0C078$_{16}$)		
set tx_enable	TxMAC_CONFIG[0]	= 1_2
(FZC_MAC+0C060$_{16}$)		

Note – The tx-output_en and tx_enable have to be programmed in the order specified.

A.9.2.10 Enable RxMAC

- **XMAC**

set rx_enable XMAC_CONFIG[8] = 1_2 (FZC_MAC+00060_{16})

- **BMAC**

set rx_enable RxMAC_CONFIG[0] = 1_2 (FZC_MAC+$0C068_{16}$)

A.9.3 IPP Initialization

A.9.3.1 HW Reset IPP

- Global hardware reset, the niu_reset_l

A.9.3.2 - Applies to all the ports, Neptune port0-3, Niu port0-1.

- Clears ipp's internal functional states, state-machines, fifo's read/write_pointers, etc.

- Clears ipp's pio accessible registers.

- Takes effect immediately not necessarily at a packet's boundary.

A.9.3.3 Soft reset IPP

Per IPP:

Assert SOFT_RST IPP_CFIG[31] = 1_2 (FZC_IPP+4000_{16})

- Clears ipp's internal functional states, state-machines, fifo's read/write_pointers, etc.

- Does NOT clear ipp's pio accessible registers.

  e.g, by writing a "1" to IPP_CFIG[31], the resulting soft reset will NOT change the IPP_CFIG[31:0] register content. If the Port's IPP was enabled, i.e., IPP_CFIG[0]=1, the IPP remains enabled since IPP_CFIG[0] will NOT be cleared by the soft reset.

- Takes effect immediately, not necessarily at a packet's boundary.

Software reset is used to reset an individual port to bring a "dead" port to life. Not required in intialization sequence.

A.9.3.4 IPP Configuration

There is NO specific initialization process/value for the IPP.

After a global reset, by default, an ipp is disabled. Sotware can further configure the IPP before enabling the IPP.

A.9.3.5 Enable IPP Port

For each port:

Assert IPP_ENABLE: IPP_CFIG[0] = 1_2. (FZC_IPP+4000_{16})

Note – The ipp_enable takes effect immediately, not necessarily at a packet boundary.Once IPP is enabled, it will issue a request to XMAC immediately, and issue an ack as a response to bMAC request right away.

A.9.3.6 Drain IPP Port

XMAC Poll IPP_DFIFO_RD_PTR[11:0] == IPP_DFIFO_RD_PTR[11:0]

BMAC Poll IPP_DFIFO_RD_PTR[10:0] == IPP_DFIFO_RD_PTR[10:0]

Note – In case of an SOP ECC error, the IPP port will not drain. A soft reset is required to reset the read/write pointers.

A.9.3.7 Disable IPP Port

For each port:

De-assert IPP_ENABLE: IPP_CFIG[0]= 0_2. (FZC_IPP+4000_{16})

Note – IPP completes receiving current packet and stops accepting new packets from MAC

Note – Immediately after an IPP port is disabled, its output data is not reliable.

Note – IPP should be disabled after MAC has been disabled

A.9.4 FFLP Initialization (classification)

A.9.4.1 Reset FFLP

- Global hardware reset, the niu_reset_l.

- Clears FFLP's internal functional states, state-machines, fifo's read/ write_pointers, etc.

- Clears ipp's pio accessible registers.

- Takes effect immediately not necessarily at a packet's boundary.

- Soft reset

FFLP does not have software reset support.

If particular IPP encounters a fatal error such as double ecc error and it requires port reset, software needs to follow mac/ipp/zcp port reset sequence. Nothing special is needed for fflp. FFLP will continue to serve other good ports.

In case if somehow ipp_fflp_dvalid stuck to 1 from one of ipp ports, fflp will continue to process pkts using whatever header sent to fflp at that time, and pass the results to zcp. FFLP does not hang and does not require reset in case there is fatal error in any ports.

A.9.4.2 Initialize FFLP

TCAM

- Disable TCAM: (After POR TCAM is ENABLED

  - Assert TCAM_DIS: FFLP_CFIG_1[26] = 1_2 (FZC_FFLP+20100_{16})

- Initialize TCAM

  - For each TCAM [Entry]:

Set:	KEY	TCAM_KEY_<0-3>[63:0]		(FZC_FFLP+20090_{16}, step 8_{16})
Set:	KEY	TCAM_KEY_MASK<0-3>[63:0]		(FZC_FFLP+$200B0_{16}$, step 8_{16})
Set:	RWC	TCAM_CTL[20:18]	= 0_{16}	(FZC_FFLP+$200D0_{16}$)
	LOC	TCAM_CTL[9:0]	= [Entry]	(FZC_FFLP+$200D0_{16}$)

  - For each TCAM associated data [Entry]:

Set:	KEY	TCAM_KEY_1[63:0]		(FZC_FFLP+20098_{16})
Set:	RWC	TCAM_CTL[20:18]	= 4_{16}	(FZC_FFLP+$200D0_{16}$)

LOC TCAM_CTL[9:0] = [Entry] (FZC_FFLP+200D0$_{16}$)

- Set TSEL

For each class:

- For each TCAM associated data [Entry]:

Set: KEY TCAM_KEY_1[11:10] = "TRES" value (FZC_FFLP+20098$_{16}$)

Set: RWC TCAM_CTL[20:18] = 4_{16} (FZC_FFLP+200D0$_{16}$)

LOC TCAM_CTL[9:0] = [Entry] (FZC_FFLP+200D0$_{16}$)

Note – In case flow table lookups are not supported, the TRES field in the TCAM associated data entries has to be set to either 01_2 or 11_2.

- Enable TCAM

- Assert TCAM_DIS: FFLP_CFIG_1[26] = 0_2 (FZC_FFLP+20100$_{16}$)

- Flow table configuration

TBD

A.9.4.3 Configure FFLP

Configure VLAN Table

for each table;

set ENET_VLAN_TBL[17:0] (FZC_FFLP+00000$_{16}$)

A.9.4.4 Enable FFLP

FFLP does not require an enable. The FFLP starts to process packets whenever a valid packet is presented by the IPP.

A.9.4.5 Disable FFLP

FFLP does not support disable. The FFLP continues to process packets whenever a valid packet is presented by the IPP.

A.9.5 RxDMA Initialization

A.9.5.1 Reset RxDMA

A.9.5.2 Hw reset RxDMA

Results in the RX_DMA POR reset state.

A.9.5.3 Stop and reset RxDMA

- De-assert EN:	RXDMA_CFIG1[31] = 0_2	(DMC+00000_{16})
- Poll QST:	RXDMA_CFIG1[29] = 1_2	indicates disable completion.
- Assert RST:	RXDMA_CFIG1[30] = 1_2	
- Poll RST:	RXDMA_CFIG1[30] = 0_2.	(HW indication of completion

---

Note – When a channel is disabled, an in progress packet is received and all packets in the RDMC pipeline will be send out prior to the disable to take affect.

RX_DMA channel is now ready for software configuration

A.9.5.4 Initialize RxDMA

- RBR Configuration

For each channel [0:15]

Set:	LEN	RBR_CFIG_A[63:48]	(DMC+00010_{16}, step 200_{16})
	STADDR_BASE	RBR_CFIG_A[43:18]	
	STADDR	RBR_CFIG_A[17:6]	
Set:	BKSIZE	RBR_CFIG_B[25:24]	(DMC+00018_{16}, step 200_{16})
	BUFSZ2	RBR_CFIG_B[17:16]	
	BUFSZ1	RBR_CFIG_B[9:8]	
	BUFSZ0	RBR_CFIG_B[1:0]	

VLD2	RBR_CFIG_B[23]	
VLD1	RBR_CFIG_B[15]	
VLD0	RBR_CFIG_B[7]	

- Mailbox configuration

For each channel [0:15]

Set:	MBADDR_H	RXDMA_CFIG1[11:0] = 1_{16}	(DMC+00000_{16}, step 200_{16})
	MBADDR_L	RXDMA_CFIG2[31:6] = 1_{16}	(DMC+00008_{16}, step 200_{16})

- RCR Configuration

For each channel [0:15]

Set:	LEN	RCR_CFIG_A[63:48]	(DMC+00040_{16}, step 200_{16})
	STADDR_BASE	RCR_CFIG_A[43:18]	
	STADDR	RCR_CFIG_A[17:6]	
Set:	PTHRES	RCR_CFIG_B[31:16]	(DMC+00048_{16}, step 200_{16})
	ENTOUT	RCR_CFIG_B[15]	
	TIMEOUT	RCR_CFIG_B[5:0]	

- Page Table Configuration

For each Channel [0-23]

Set	FUNC	RX_LOG_PAGE_VLD[3:2]	(DMC+20000_{16}, step 0x200)
	PAGE1	RX_LOG_PAGE_VLD[1]	
	PAGE0	RX_LOG_PAGE_VLD[0]	
Set	MASK	RX_LOG_MASK<1-2>[31:0]	(DMC+20008_{16}, step 200_{16})
Set	VALUE	RX_LOG_VALUE<1-2>[31:0]	(DMC+20010_{16}, step 200_{16})
Set	RELO	RX_LOG_PAGE_RELO<1-2>[31:0]	(DMC+20028_{16}, step 200_{16})
Set	HANDLE	RX_LOG_PAGE_HDL<1-2>[19:0]	(DMC+20038_{16}, step 200_{16})

Note – The page table configuration registers can only be changed if the DMA channel is disabled and reset.

A.9.5.5 Enable RxDMA

For each channel:

- Assert EN: RXDMA_CFIG1[31] = 1_2 (DMC+00000_{16})

RX_DMA channel is now ready to receive packets.

A.9.5.6 Disable RxDMA

For each channel:

- De-assert EN: RXDMA_CFIG1[31] = 0_2. (DMC+00000_{16})

- Poll asserts QST completion RXDMA_CFIG1[30] = 1_2 indicates disable

---

Note – When a channel is disabled, an in progress packet is received and all packets in the RDMC pipeline will be send out prior to the disable to take affect.

A.9.6 TxDMA Initialization

A.9.6.1 HW Reset TxDMA

- Global hardware reset, the niu_reset_l

Results in the TX_DMA POR reset state:

RST_STATE	TX_CS[30] = 1_2 (in reset)	[DMC+40028_{16}]
TXC_ENABLED	TXC_CONTROL = 0_2 (disabled)	[DMC+20000_{16}]

A.9.6.2 Soft Reset TxDMA

Tx DMA channle should be reset only if that DMA channel is already bound to a port in TXC. In the absence of this depending upon the state of the DMA, it may never reset.

- Assert RST:	TX_CS[31] = 1_2	[DMC+40028_{16}]
- De-assert STOP_EN_GO	TX_CS[28] = 0_2	[DMC+40028_{16}]
- Poll RST_STATE	TX_CS[30] = 1_2	[DMC+40028_{16}]

A.9.6.3 Stop TxDMA

Tx DMA channel should be stopped only if that DMA is already bound to a port in TXC. In the absence of this depending upon the state of the DMA, it may be never stopt.

- Assert STOP_N_GO:	TX_CS[28] = 1_2	[DMC+40028_{16}]
- Poll SNG_STATE	TX_CS[27] = 1_2	[DMC+40028_{16}]

A.9.6.4 Stop and Reset TxDMA.

- Apply Stop followed by Soft Reset sequence

A.9.6.5 Configure TXC

- Enable TXC ports:

  - Set PORT<0-3>_ENABLED: TXC_CONTROL[3:0] = 1_{16} (FZC_TXC+20000_{16})

A.9.6.6 Enable TXC

- Enable TXC controller

  - Assert TXC_ENABLED: TXC_CONTROL[4] = 1_{12}(FZC_TXC+20000_{16})

Note – Configure and enable of TXC can be done simultaniously

---

A.9.6.7 Bind TxDMA channel to Port

For each DMA channel [x]:

- Set PORT_DMA_LIST<0-3> TXC_PORT_DMA[x] = 1_2 (FZC_TXC+20028_{16})
- Set DMA_MAX_BURST TXCDMA_MAX[19:0] (FZC_TXC+20000_{16})

Note – A DMA hannel can be added/removed during run time using PIO read-mod-write.

Note – A DMA channel can only be bound to one port. Binding a channel to multiple ports results in undesirable side effects.

A.9.6.8 Initialize TxDMA

- Transmit ring configuration

For each Channel [0-23]

Set	LEN	TX_RNG_CFIG[60:48]	(DMC+40000_{16}, step 200_{16})
	STADDR_BASE	TX_RNG_CFIG[43:19]	
	STADDR	TX_RNG_CFIG[18:6]	

- Mailbox config

For each Channel [0-23]

Set	MBADDR	TXDMA_MBH[43:19]	(DMC+40030_{16}, step 200_{16})
Set	MBADDR	TXDMA_MBL[31:6]	(DMC+40038_{16}, step 200_{16})

- Page Table Configuration

For each Channel [0-23]

Set	FUNC	TX_LOG_PAGE_VLD[3:2]	(DMC+40000_{16}, step 200_{16})
	PAGE1	TX_LOG_PAGE_VLD[1]	
	PAGE0	TX_LOG_PAGE_VLD[0]	
Set	MASK	TX_LOG_MASK<1-2>[31:0]	(DMC+40008_{16}, step 200_{16})
Set	VALUE	TX_LOG_VALUE<1-2>[31:0]	(DMC+40010_{16}, step 200_{16})
Set	RELO	TX_LOG_PAGE_RELO<1-2>[31:0]	(DMC+40028_{16}, step 200_{16})
Set	HANDLE	TX_LOG_PAGE_HDL<1-2>[19:0]	(DMC+40038_{16}, step 200_{16})

Note – The page table configuration registers can only be changed if the Tx DMA channel is disabled.

- Clear tail register

For each Channel [0-23]

Set	WRAP	TX_RING_KICK[19] = 0_{16}	(DMC+40018_{16}, step 200_{16})

Set TAIL TX_RING_KICK[18:3] = 0000_{16}

A.9.6.9 Enable TxDMA

For each channel [0-23]:

De-assert RST_STATE TX_CS[30] = 0_2 (DMC+40028_{16}, step 200_{16})

Note – Note: TX_DMA channel is now ready to transmit packets.

A.9.6.10 Disable TxDMA

For each channel [0-23]:

Stop:	- Assert STOP_N_GO:	TX_CS[28] = 1_2	(DMC+40028_{16}, step 200_{16})
Poll:	- SNG_STATE	TX_CS[27] = 1_2	(DMC+40028_{16}, step 200_{16})
Reset:	- Assert RST:	TX_CS[31] = 1_2	(DMC+40028_{16}, step 200_{16})
	- De-assert STOP_EN_GO	TX_CS[28] = 0_2	(DMC+40028_{16}, step 200_{16})
Poll	- RST_STATE	TX_CS[30] = 1_2	(DMC+40028_{16}, step 200_{16})

Note – Channel is disabled at a packet boundary. Transmit of packets in progress are completed before disable becomes active.

Note – TX controller and TxMAC need to be enabled when TX_DMA channels are disabled. Ethernet Transmit link has to be up before TxDMA can be disabled.

A.9.6.11 Disable TXC ports

- De-allocate Ports

- Reset PORT<0-3>_ENABLED: TXC_CONTROL[3:0] = 1_{16}(FZC_TXC+20000_{16})

A.9.6.12 Disable TXC controller

- Disable TXC controller

- De-assert TXC_ENABLED: TXC_CONTROL[4] = 0_2.(FZC_TXC+20000_{16})

Note – The TXC is not expected to be disabled at runtime. For debug purpose only

A.9.7 ZCP Initialization

There is no POR init sequence needed for current ZCP bypass mode.

Default ZC_ENABLE ZCP_CFIG[0] = 0_2 (FZC_ZCP+00000_{16})

Note – The current Neptune did not fully verify the ZCP feature.

A.9.7.1 Reset ZCP Control Fifo

For each of the four ports:

Set RST_CFIFO3 RESET_CFIFO[3] = 1_2 (FZC_ZCP+00098_{16})

Set RST_CFIFO2 RESET_CFIFO[2] = 1_2

Set RST_CFIFO1 RESET_CFIFO[1] = 1_2

Set RST_CFIFO0 RESET_CFIFO[0] = 1_2

Note – Set bits to 0_{16} to de-assert reset

A.9.7.2 Configure ZCP-RDC table

For each of 128 entries

Set RDC RDC_TABLE[3:0] = Ch# (FZC_ZCP+10000_{16})

APPENDIX **B**

Neptune Configuration Overview

B.1 Port Configuration Summary

TABLE 0-1 Neptune MAC Port Configurations *(1 of 3)*

Configuration	Port0	Port1	Port2	Port3	Mode Setting
2x10G (XAUI) + 2x1G (RGMII)	10G (XAUI) (PLL0)	10G (XAUI) (PLL1)	1G (RGMII)	1G (RGMII)	xgmii_mode0 = 1 gmii_mode0 = 0 mii_mode0 = 0 pcs_bypass0 = 0 (optical) xgmii_mode1 = 1 gmii_mode1 = 0 mii_mode1 = 0 pcs_bypass1 = 0 (optical) ---------- gmii_mode2 = 1 phy_mode2 = 1 gmii_mode3 = 1 phy_mode3 = 1 ---------- atca_ge = 0 (mif)
Programming steps: **1**. addr= FZC_MAC + x00060, bit[30] = 0 (pcs_bypass), bit [28:27] = 0 (xgmii_mode0), **2**. addr= FZC_MAC + x06060, bit[30] = 0 (pcs_bypass), bit [28:27] = 0 (xgmii_mode1), **3**. addr= FZC_MAC + x0C078, bit[3] = 1 (gmii_mode2), **4**. addr= FZC_MAC + x0E000, bit[1] = 1 (phy_mode2), **5**. addr= FZC_MAC + x10078, bit[3] = 1 (gmii_mode3), **6**. addr= FZC_MAC + x12000, bit[1] = 1 (phy_mode3), **7**. addr= FZC_MAC + x16000, bit[16] = 0 (atca_GE).					

TABLE 0-1 Neptune MAC Port Configurations *(2 of 3)*

Configuration	Port0	Port1	Port2	Port3	Mode Setting
1x10G (XAUI) + 3x1G (RGMII)	10G (XAUI)	1G (RGMII)	1G (RGMII)	1G (RGMII)	xgmii_mode0 = 1 gmii_mode0 = 0 mii_mode0 = 0 pcs_bypass0 = 0 (optical) xgmii_mode1 = 0 gmii_mode1 = 1 mii_mode1 = 0 pcs_bypass1 = 1 (copper) --- gmii_mode2 = 1 phy_mode2 = 1 gmii_mode3 = 1 phy_mode3 = 1 --- atca_ge = 0 (mif)
Programming steps: **1**. addr= FZC_MAC + x00060, bit[30] = 0 (pcs_bypass), bit [28:27] = 0 (xgmii_mode0), **2**. addr= FZC_MAC + x06060, bit[30] = 1 (pcs_bypass), bit [28:27] = 01 (gmii_mode1), **3**. addr= FZC_MAC + x0C078, bit[3] = 1 (gmii_mode2), **4**. addr= FZC_MAC + x0E000, bit[1] = 1 (phy_mode2), **5**. addr= FZC_MAC + x10078, bit[3] = 1 (gmii_mode3), **6**. addr= FZC_MAC + x12000, bit[1] = 1 (phy_mode3), **7**. addr= FZC_MAC + x16000, bit[16] = 0 (atca_GE).					
4x1G (RGMII)	1G (RGMII)	1G (RGMII)	1G (RGMII)	1G (RGMII)	xgmii_mode0 = 0 gmii_mode0 = 1 mii_mode0 = 0 pcs_bypass0 = 1 (copper) xgmii_mode1 = 0 gmii_mode1 = 1 mii_mode1 = 0 pcs_bypass1 = 1 (copper) --- gmii_mode2 = 1 phy_mode2 = 1 gmii_mode3 = 1 phy_mode3 = 1 --- atca_ge = 0 (mif)
Programming steps: **1**. addr= FZC_MAC + x00060, bit[30] = 1 (pcs_bypass), bit [28:27] = 01 (gmii_mode0), **2**. addr= FZC_MAC + x06060, bit[30] = 1 (pcs_bypass), bit [28:27] = 01 (gmii_mode1), **3**. addr= FZC_MAC + x0C078, bit[3] = 1 (gmii_mode2), **4**. addr= FZC_MAC + x0E000, bit[1] = 1 (phy_mode2), **5**. addr= FZC_MAC + x10078, bit[3] = 1 (gmii_mode3), **6**. addr= FZC_MAC + x12000, bit[1] = 1 (phy_mode3), **7**. addr= FZC_MAC + x16000, bit[16] = 0 (atca_GE).					

TABLE 0-1 Neptune MAC Port Configurations *(3 of 3)*

Configuration	Port0	Port1	Port2	Port3	Mode Setting
2x1G (optical) + 2x1G(RGMII) ATCA_GE mode	Lane 0, 1G (optical) (PLL0)	Lane 0, 1G (optical) (PLL1)	1G (RGMII)	1G (RGMII)	xgmii_mode0 = 0 gmii_mode0 = 1 mii_mode0 = 0 pcs_bypass0 = 0 (optical) xgmii_mode1 = 0 gmii_mode1 = 1 mii_mode1 = 0 pcs_bypass1 = 0 (optical) ---------- gmii_mode2 = 1 phy_mode2 = 1 gmii_mode3 = 1 phy_mode3 = 1 ---------- atca_ge = 1 (mif) Notes: the function of phy_mode in bmac is equivalent to pcs_bypass in xmac.
Programming steps: **1**. addr= FZC_MAC + x00060, bit[30] = 0 (pcs_bypass), bit [28:27] = 01 (gmii_mode0), **2**. addr= FZC_MAC + x06060, bit[30] = 0 (pcs_bypass), bit [28:27] = 01 (gmii_mode1), **3**. addr= FZC_MAC + x0C078, bit[3] = 1 (gmii_mode2), **4**. addr= FZC_MAC + x0E000, bit[1] = 1 (phy_mode2), **5**. addr= FZC_MAC + x10078, bit[3] = 1 (gmii_mode3), **6**. addr= FZC_MAC + x12000, bit[1] = 1 (phy_mode3), **7**. addr= FZC_MAC + x16000, bit[16] = 1 (atca_GE).					